A Final Report
Grant No. N00014-95-1-0208
November 1, 1995 - December 31, 1996

TIMELINESS AND PREDICTABILITY IN REAL-TIME DATABASE SYSTEMS

Submitted to:

Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217-5660

Attention: Dr. Gary M. Koob, Code 311

Submitted by:

Sang H. Son
Associate Professor

SEAS Report No. UVA/525499/CS97/101
February 1997

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF
ENGINEERING
& APPLIED SCIENCE

University of Virginia
Thornton Hall
Charlottesville, VA 22903

M97-06-3663

UNIVERSITY OF VIRGINIA
School of Engineering and Applied Science

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 600. There are 160 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Applied Mechanics, Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 17,000), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Phyics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.

AQ #M97-06-3663

PLEASE CHECK THE APPROPRIATE BLOCK BELOW:

☐ _____ copies are being forwarded. Indicate whether Statement A, B, C, D, E, F, or X applies.

☒ DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

☐ DISTRIBUTION STATEMENT B:
DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES ONLY; (Indicate Reason and Date). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED TO (Indicate Controlling DoD Office).

☐ DISTRIBUTION STATEMENT C:
DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND THEIR CONTRACTORS; (Indicate Reason and Date). OTHER REQUESTS FOR THIS DOCUMENT SHALL BE REFERRED TO (Indicate Controlling DoD Office).

☐ DISTRIBUTION STATEMENT D:
DISTRIBUTION AUTHORIZED TO DoD AND U.S. DoD CONTRACTORS ONLY; (Indicate Reason and Date). OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office).

☐ DISTRIBUTION STATEMENT E:
DISTRIBUTION AUTHORIZED TO DoD COMPONENTS ONLY; (Indicate Reason and Date). OTHER REQUESTS SHALL BE REFERRED TO (Indicate Controlling DoD Office).

☐ DISTRIBUTION STATEMENT F:
FURTHER DISSEMINATION ONLY AS DIRECTED BY (Indicate Controlling DoD Office and Date) or HIGHER DoD AUTHORITY.

☐ DISTRIBUTION STATEMENT X:
DISTRIBUTION AUTHORIZED TO U.S. GOVERNMENT AGENCIES AND PRIVATE INDIVIDUALS OR ENTERPRISES ELIGIBLE TO OBTAIN EXPORT-CONTROLLED TECHNICAL DATA IN ACCORDANCE WITH DoD DIRECTIVE 5230.25, WITHHOLDING OF UNCLASSIFIED TECHNICAL DATA FROM PUBLIC DISCLOSURE, 6 Nov 1984 (Indicate date of determination). CONTROLLING DoD OFFICE IS (Indicate Controlling DoD Office).

☐ This document was previously forwarded to DTIC on _____ (date) and the AD number is _____.

☐ In accordance with provisions of DoD instructions, the document requested is not supplied because:

☐ It will be published at a later date. (Enter approximate date, if known).

☐ Other. (Give Reason)

DoD Directive 5230.24, "Distribution Statements on Technical Documents," 18 Mar 87, contains seven distribution statements, as described briefly above. Technical Documents must be assigned distribution statements.

Per telecon with A. Van T./ROR 3-18-97
Authorized Signature/Date

Print or Type Name

696-4312
Telephone Number

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	February 1997	Final 1/1/95 - 12/31/1996

4. TITLE AND SUBTITLE
Timeliness and Predictability in Real-Time Database Systems

5. FUNDING NUMBERS
Grant No. N00014-95-1-0208

6. AUTHORS(S)
Sang H. Son

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)
University of Virginia
Department of Computer Science
School of Engineering and Applied Science
Thornton Hall
Charlottesville, VA 22903-2442

8. PERFORMING ORGANIZATION REPORT NUMBER
UVA/525499/CS97/101

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The view, opinion, and/or findings contained in the report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

See report.

14. SUBJECT TERMS

15. NUMBER OF PAGES
7

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Timeliness and Predictability in Real-Time Database Systems

Sang H. Son

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

Contract Information

Timeliness and Predictability in Real-Time Database Systems
N00014-95-1-0208
Sang H. Son
(804) 982-2205
son@cs.virginia.edu
1/1/95 - 12/31/96

1. Research Objectives

The confluence of computers, communications, and databases is quickly creating a globally distributed database where many applications require real-time access to both temporally accurate and multimedia data. This is particularly true in military and intelligence applications, but these required features are needed in many commercial applications as well. Mjor applications are military command and control, avionics and weapon systems (e.g., missile guidance system), and monitoring and decision support systems. Those applications have at their core requirements for managing and analyzing massive amounts of data residing in many data repositories. Much of this data has timing attributes such as a particular satellite image being valid for no more than 5 minutes. Audio, video and images are key types of data which provide increased value to applications, but also increased challenges. Driving such systems are significant real-time requirements for managing thousands of objects and tracking them by using a global, intelligent, and responsive multimedia database system. They have the following characteristics:

(1) transactions with timing constraints
(2) data with temporal properties
(3) distributed multimedia data
(4) mixture of sensitive and unclassified data

Those characteristics lead to the following requirements:

(1) timeliness and predictability
(2) temporal consistency
(3) integrated support of soft, firm, and hard deadlines
(4) storage, retrieval and synchronization of multimedia data
(5) security enforcement

(6) high reliability

(7) scalability of solutions

The objective of this project was to develop new database system technology for distributed real-time systems and to evaluate them in the experimental real-time database servers. Our focus has been to discover a set of design principles for building dependable and responsive database systems for time-critical applications and to develop algorithms to improve timeliness and predictability of such systems.

2. Technical Approach

Our approach to achieving the objectives stated above has been four-fold:

(1) develop a database and transaction model

(2) design scheduling algorithms to support timely, secure, and predictable execution of transactions

(3) develop paradigms and mechanisms for constructing real-time database systems

(4) evaluate the performance of the algorithms and mechanisms developed in this project using simulation as well as experimental real-time database systems.

The first two approaches (modelling and algorithm design) can be considered as basic research for setting up foundations in this area. The other two deal with actual application of the technology developed to realistic situations to evaluate their merits. Research results coming out from the second approach (practical issues) are then used to revise the models and algorithms. This kind of feedback loop approach turns out to be very effective in performing research in real-time systems area.

3. Accomplishments

3.1. Schemes for predictable transaction processing

There are many cases of hard real-time database applications in real world, such as flight control systems and missile guidance systems, and thus a flexible real-time database management system must provide mechanisms to minimize the execution time variance of a transaction, making the system's behavior predictable. We provide a framework to classify different types of transactions and to develop processing schemes for each type of transactions that can be supported in predictable real-time database systems.

3.2. New scheduling and concurrency control algorithms

The algorithms we developed based on the notion of dynamic adjustment of serialization order reduce unnecessary blocking and aborts, significantly improving the timeliness of transactions. While the original priority ceiling protocol provides a bound on transaction blocking delay and schedulability analysis, they often suffer from the problem of unnecessary blockings due to its conservative scheduling policy. The main reason for their conservatism is the implicit assumption that if a transaction conflicts with other executing transaction, it is unable to preempt the conflicting transaction. We have shown that real-time database systems could avoid unnecessary blockings using the notion of dynamic adjustment of serialization order, resulting in improved performance. Our results will be presented at the 13th IEEE Conference on Data Engineering, to be held in England in April 1997.

3.3. Integrating security with real-time requirements

In principle, any system that maintains sensitive information to be shared by multiple applications with different levels of security clearance requires multilevel security. Many of those systems also require real-time database accesses. Especially for DoD/ Navy applications, security is considered

essential, in addition to real-time requirements. It has been shown that most conventional schedulers are not satisfying the security requirements. It is necessary to develop new algorithms for scheduling and concurrency control that can ensure security according to appropriate models for secure real-time databases. Our algorithms are the first ones that attempt to support both security and real-time requirements.

3.4. Multimedia data management

The essential problem of multimedia is not of providing support for individual media, rather support for "synchronizing" the otherwise autonomous data transfers across computers. The issue of real-time and synchronization mechanisms become even more challenging in distributed systems as the different media streams may arrive from different sources. Our research resulted in a systematic scheme to specify and enforce synchronization requirements in real-time systems.

3.5. Fault-tolerant scheduling algorithms

Existing fault tolerance techniques for transaction systems are inadequate in a distributed real-time system environment with respect to assuring system responsiveness - that no processing, communications, and database transaction deadlines will be missed. Since missing a transaction deadline can be catastrophic in some circumstances, the fault-tolerance of these systems must be addressed. We developed several algorithms to guarantee the hard deadlines of transactions on a multiprocessor system through statically scheduling the transactions onto different processors and using a recovery block approach to tolerate processor failures.

3.6. Replication control for critical information

As in any other systems, critical data should be replicated in real-time database systems for improved availability and fault-tolerance. However, replication involves certain overhead to maintain replicated copies in consistent states, and conventional one-copy serializability might not be desirable for its high overhead. We have developed replication management algorithms, based on a weaker correctness criterion called epsilon-serializability.

3.7. Development of experimental real-time database servers

Previous work in real-time database systems has primarily based on simulation. Our research has focused on how current real-time technology can be applied to architect an actual real-time database system. A real real-time database system must confront many practical issues which simulation studies typically ignore: race conditions, concurrency, and asynchrony. By actually constructing a real-time database server on top of several platforms including real-time kernels, we have identified many implementation issues and developed practical solutions to those problems.

Honors and Recognition

- General Chair, 18th IEEE Real-Time Systems Symposium, 1997.
- Program Co-Chair, Fourth International Workshop on Real-Time Computing Systems and Applications, 1997.
- General Chair, 2nd International Workshop on Real-Time Database Systems, 1997.
- Program Chair, 17th IEEE Real-Time Systems Symposium, 1996.
- Program Chair, International Workshop on Real-Time Database Systems, 1996.

- Program Vice-Chair, 4th IEEE Workshop on Parallel and Distributed Real-Time Systems, 1996.
- Tutorial Chair, 3rd International Workshop on Real-Time Computing Systems and Applications, 1996.
- Program Committee, IEEE Real-Time Technology and Applications Symposium, 1997.
- Program Committee, ACM Workshop on Databases: Active and Real-Time (DART'96), 1996.
- Program Committee, International Conference on Information and Knowledge Management, 1996.
- Program Committee, International Workshop on Active Real-Time Database Systems (ARTDB), 1995, 1997.
- Program Committee, IEEE Workshop on Future Trends of Distributed Computing Systems, 1995.
- Program Committee, International Conference on Intelligent Information Management Systems, 1995, 1996.
- Program Committee, IEEE International Conference on Data Engineering, 1995.
- Program Committee, IEEE Workshop on Parallel and Distributed Real-Time Systems, 1995, 1996, 1997.
- Program Committee, Euromicro Workshop on Real-Time Systems, 1995, 1996.
- Program Committee, IEEE Symposium on Real-Time Systems, 1995.

Publications

- **Books and Book Chapters**

(1) S. H. Son, K. J. Lin, and A. Bestavros (eds.), *Real-Time Database Systems: Issues and Applications,* Kluwer Academic Publishers, 1997 (to appear).

(2) S. H. Son (ed.), *Advances in Real-Time Systems,* Prentice Hall, 1995.

(3) A. Bestavros, K. Lin, and S. H. Son, "Advances in Real-Time Database Systems Research," *Real-Time Database Systems: Issues and Applications,* S. H. Son, K. J. Lin, and A. Bestavros (eds.), Kluwer Academic Publishers, 1997 (to appear).

(4) Y. Kim and S. H. Son, "Developing a Real-Time Database: The StarBase Experience," *Real-Time Database Systems: Issues and Applications,* S. H. Son, K. J. Lin, and A. Bestavros (eds.), Kluwer Academic Publishers, 1997 (to appear).

(5) S. H. Son, R. David, and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems," *Database Security: Status and Prospects,* D. Spooner, S. Demurjian, and J. Dobson (eds.), Chapman and Hall Publishing, pp 199-214, 1996.

(6) R. Mukkamala and S. H. Son, "A Secure Concurrency Control Protocol for Real-Time Databases," *Database Security: Status and Prospects,* D. Spooner, S. Demurjian, and J. Dobson (eds.), Chapman and Hall Publishing, pp 215-230, 1996.

(7) J. Lee and S. H. Son, "Concurrency Control Algorithms for Real-Time Database Systems," *Performance of Concurrency Control Mechanisms in Centralized Database Systems,* V. Kumar (ed.), Prentice Hall, pp 429-460, 1996.

(8) Y. Oh and S. H. Son, "Enhancing Fault Tolerance in Rate-Monotonic Scheduling," *Responsive Computing,* M. Malek (ed.), Kluwer Academic Publishers, 1995.

(9) Y. Kim and S. H. Son, "Predictability and Consistency in Real-Time Database Systems," *Advances in Real-Time Systems,* S. H. Son (ed.), Prentice Hall, pp 509-531, 1995.

• Journal Publications

(1) K. Lam, S. Hung, and S. H. Son, "On Using Real-Time Static Locking Protocols for Distributed Real-Time Databases," *Journal of Real-Time Systems,* (accepted).

(2) Y. K. Kim, M. Lehr, and S. H. Son, "Software Architecture for a Firm Real-Time Database System," *Journal of Systems Architecture, Special Issue on Real-Time Systems,* (accepted).

(3) S. H. Son, R. David, and C. Chaney, "Design and Analysis of an Adaptive Policy for Secure Real-Time Locking Protocol," *Journal of Information Sciences,* vol. 99, June 1997 (to appear).

(4) N. Agarwal and S. H. Son, "A Model for Specification and Synchronization of Data for Distributed Multimedia Applications," *Journal of Multimedia Tools and Applications,* vol. 3, no. 2, pp 79-104, Sept. 1996.

(5) W. Cho, C. Park, K. Whang, and S. H. Son, "A New Method for Estimating the Number of Objects Satisfying an Object-Oriented Query Involving Partial Participation of Classes," *Information Systems,* vol. 21, no. 3, pp 253-267, June 1996.

(6) S. H. Son, F. Zhang, and B. Hwang, "Concurrency Control for Replicated Data in Distributed Real-Time Systems," *Journal of Database Management, Special Issue on Real-time Database Systems: Theory and Practice,* vol. 7, no. 2, pp 12-23, March 1996.

(7) A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems," *IEEE Transactions on Computers,* vol. 44, no. 12, pp 1429-1442, December 1995.

(8) Y. Oh and S. H. Son, "Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems," *Journal of Real-Time Systems,* vol. 9, no. 3, pp 207-239, Sept. 1995.

(9) S. H. Son, J. Ratner and S. Chiang, "StarBase: A Simulation Laboratory for Distributed Database Research," *Journal of Computer Simulation,* vol. 5, no. 3, pp 327-350, 1995.

(10) S. H. Son and S. Park, "A Priority-Based Scheduling Algorithm for Real-Time Databases," *Journal of Information Science and Engineering,* vol. 11, no. 2, pp 233-248, June 1995.

(11) Y. Kim, M. Lehr, D. George, and S. H. Son, "Supporting Real-Time Transactions in Distributed Time-Critical Applications: Issues and Experiences," *Journal of Mini and Microcomputers, Special Issue on Parallel and Distributed Real-Time Systems,* vol. 17, no. 2, pp 63-70, 1995.

(12) J. Sklenar, N. Agarwal, J. Dent, S. H. Son, and S. Kaul, "Compression of Two-Dimensional Echocardiographic Images: How Far Can We Go?" *Cardiovascular Imaging,* vol. 7, pp 49 - 53, 1995.

- **Conference Publications**

(1) S. H. Son, "Supporting Timeliness and Security in Real-Time Database Systems," *9th Euromicro Workshop on Real-Time Systems,* Toledo, Spain, June 1997 (to appear).

(2) J. Taina and S. H. Son, "Requirements for Real-Time Object-Oriented Database Models - How Much is Too Much?" *9th Euromicro Workshop on Real-Time Systems,* Toledo, Spain, June 1997 (to appear).

(3) K. Lam, S. H. Son, and S. Hung, "A Priority Ceiling Protocol with Dynamic Adjustment of Serialization Order," *13th IEEE Conference on Data Engineering,* Birmingham, UK, April 1997 (to appear).

(4) J. Taina and S. H. Son, "A Framework for Real-Time Object-Oriented Database Models," *3rd IEEE Workshop on Object-oriented Real-time Dependable Systems,* Newport Beach, CA, Feb. 1997.

(5) C. G. Lee, S. H. Son, S. Min, and C. Kim, "Efficiently Supporting Hard/Soft Deadline Transactions in Real-Time Database Systems," *3rd International Workshop on Real-Time Computing Systems and Applications,* Seoul, Korea, Oct. 1996, pp 74-80.

(6) B. Hwang and S. H. Son, "Decentralized Transaction Management in Multidatabase Systems," *20th International Computer Software and Applications Conference (COMPSAC'96),* Seoul, Korea, August 1996, pp 192-198.

(7) Y. Kim, and S. H. Son, "Supporting Predictability in Real-Time Database Systems," *IEEE Real-Time Technology and Applications Symposium (RTAS'96),* Boston, MA, June 1996, pp 38-48.

(8) M. Lehr, Y. Kim, and S. H. Son, "Managing Contention and Timing Constraints in a Real-Time Database System," *16th IEEE Real-Time Systems Symposium,* Pisa, Italy, Dec. 1995, pp 332-341.

(9) S. Shih, Y. Kim, and S. H. Son, "Performance Evaluation of a Firm Real-Time Database System," *2nd International Workshop on Real-Time Computing Systems and Applications,* Tokyo, Japan, Oct. 1995, pp 116-124.

(10) S. H. Son, R. David, and B. Thuraisingham, "An Adaptive Policy for Improved Timeliness in Secure Database Systems," *Annual IFIP WG 11.3 Conference of Database Security,* Rensselaerville, New York, Aug. 1995, pp 223-233.

(11) R. Mukkamala and S. H. Son, "A Secure Concurrency Control Protocol for Real-Time Databases," *Annual IFIP WG 11.3 Conference of Database Security,* Rensselaerville, New York, Aug. 1995, pp 235-253.

(12) S. H. Son, "System Issues in Supporting Active Real-Time Databases," *International Workshop on Active Real-Time Database Systems (ARTDB'95),* Skovde, Sweden, June 1995, Lecture Notes in Computer Science, Springer-Verlag.

(13) M. Lehr, Y. Kim, and S. H. Son, "StarBase: A Firm Real-Time Database Manager for Time-Critical Applications," *7th Euromicro Workshop on Real-Time Systems,* Odense, Denmark, June 1995, pp 317-322.

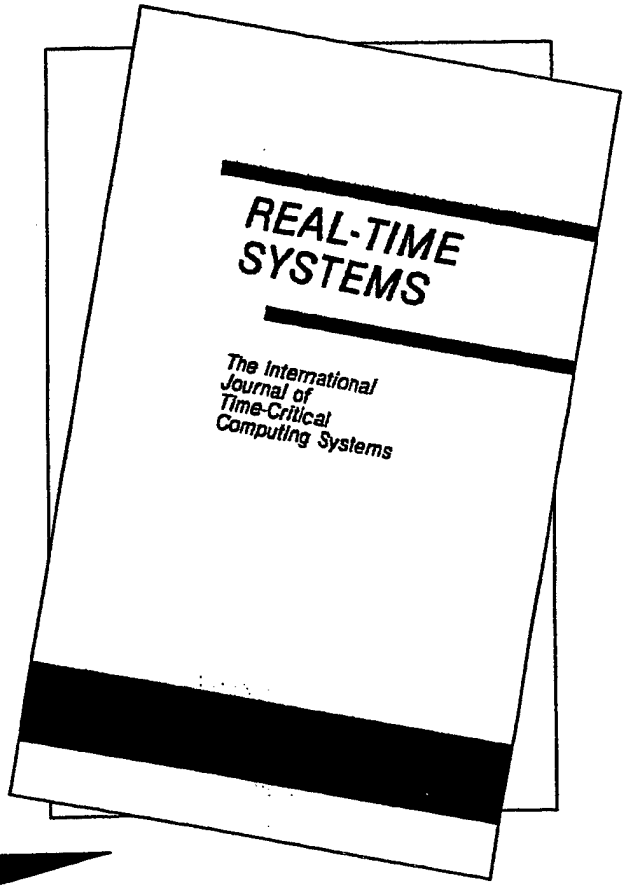
(14) S. H. Son, R. David, and R. Mukkamala, "Supporting Security Requirements in Multilevel Real-Time Databases," *IEEE Symposium on Security and Privacy,* Oakland, CA, May 1995, pp 199-210.

(15) S. H. Son and F. Zhang, "Real-Time Replication Control for Distributed Database Systems: Algorithms and Their Performance," *The Fourth International Symposium on Database Systems for Advanced Applications (DASFAA '95),* Singapore, April 1995, pp 214-221.

APPENDIX

ISSN 0922-6443

OFFPRINT FROM



Kluwer academic publishers

Real-Time Systems

The International Journal of Time-Critical Computing Systems

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced. The journal *Real-Time Systems* publishes papers that concentrate on real-time computing principles and applications. *Real-Time Systems* publishes research papers, invited papers, project reports and case studies, standards and corresponding proposals for general discussion, and a partitioned tutorial on real-time systems as a continuing series. Much of the work in building sophisticated, modern real-time systems is interdisciplinary in nature and, up until now, has been found scattered throughout the primary literature. *Real-Time Systems* provides a single-source coverage of the state of the art in this exciting and expanding field. The editorial board, the writers and the readers of the journal are drawn from all parts of the world, from industry, and from academe. Papers published in *Real-Time Systems* cover the following topics, among others: requirements engineering, specification and verification techniques, design methods and tools, programming languages, operating systems, scheduling algorithms, architecture and hardware (especially interfacing), fault tolerance, distributed and other novel architectures, communications, distributed databases, AI, expert systems, and application case studies. Applications are found in command and control systems, process control systems, automated manufacturing, flight control, avionics, space avionics and defense systems, shipborne systems, vision and robotics, and robot teams in hazardous environments.

Real-Time Systems is abstracted/indexed in: *Engineering Index; INSPEC Information Services; ISI: CompuMath Citation Index; Information Science Abstracts; ACM Guide to Computing Literature; ACM Computing Reviews; COMPENDEX*Plus database; ISI: SciSearch; ISI: Automatic Subject Citation Alert; Research Alert; Current Contents/Engineering, Technology, & Applied Sciences; Zentralblatt für Mathematik; Computer Literature Index; Information Science Abstracts; Psychological Abstracts*

'Real-Time Systems *does what its title suggests, which is to provide a welcome service in drawing together material that is otherwise not easy to find in one place.* Real-Time Systems *has a good chance of being the main focus in print for the future development of its subject; both developers and users stand to benefit from reading it.'* **Nature,** Vol. 347, Oct. 1990

Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, MA 02061, U.S.A.

Real-Time Systems, 9, 207–239 (1995)

Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems

YINGFENG OH AND SANG H. SON {yo5u, son}@cs.virginia.edu
Department of Computer Science, University of Virginia, Charlottesville, VA 22903, USA

Abstract. In this paper, we study the problem of allocating a set of periodic tasks on a multiprocessor system such that tasks are scheduled to meet their deadlines on individual processors by the Rate-Monotonic scheduling algorithm. A new schedulability condition is developed for the Rate-Monotonic scheduling that allows us to develop more efficient on-line allocation algorithms. Two on-line allocation algorithms—RM-FF and RM-BF are presented, and shown that their worst-case performance, over the optimal allocation, is upper bounded by 2.33 and lower bounded by 2.28. Then RM-FF and RM-BF are further improved to form two new algorithms: Refined-RM-FF (RRM-FF) and Refined-RM-BF (RRM-BF), both of which have a worst-case performance bound of 2. We also show that when the maximum allowable utilization of a task is small, the worst-case performance of all the new algorithms can be significantly improved. The worst-case performance bounds of RRM-FF and RRM-BF are currently the best bounds in the class of on-line scheduling algorithms proposed to solve the same scheduling problem. Simulation studies show that the average-case performance of the newly proposed algorithms is significantly superior to those in the existing literature.

1. Introduction

Rate-Monotonic (RM) scheduling (Liu and Layland, 1973) is becoming a viable scheduling technique for real-time systems. Through the years, researchers have successfully applied this technique to tackle a number of practical problems, such as task synchronization, bus scheduling, joint scheduling of periodic and aperiodic tasks, and transient overload (Joseph and Pandya, 1986), (Lehoczky et al., 1989), (Lehoczky, 1992), (Ramos-Thuel and Stronider, 1991), (Sha et al., 1986), (Rajkumar, 1989), (Rajkumar and Lehoczky, 1990), (Rajkumar and Lehoczky, 1990), (Sha and Goodenough, 1990), (Sha and Lehoczky, 1989), (Tindell et al., 1992). In each of these areas, conventional rate-monotonic scheduling has been adapted and extended to produce effective scheduling algorithms.

While rate-monotonic scheduling is optimal for uniprocessor systems with fixed-priority assignment, it is, unfortunately, not so for multiprocessor systems. In fact, the problem of optimally scheduling a set of periodic tasks on a multiprocessor system using either fixed-priority or dynamic priority assignment is known to be NP-complete (Leung and Whitehead, 1982). Hence, any practical solution to the problem of scheduling real-time tasks on multiprocessor systems presents a trade-off between computational complexity and performance. Heuristic algorithms have been shown to deliver near-optimal solutions with limited computational overhead.

In this study, we are concerned with developing efficient heuristic algorithms for scheduling a set of periodic tasks on a multiprocessor system. The general solution to such a problem involves two algorithms: one to assign tasks to processors, and the other to schedule tasks assigned on each individual processor. Two major strategies exist

for assigning tasks to processors: partitioning and non-partitioning strategies. In a non-partitioning strategy, each occurrence of a task may be executed on a different processor, while a partitioning strategy requires that all occurrences of a task must be executed on the same processor. The partitioning strategy is often preferred, because relatively low overhead is involved in the scheduling process. A scheduling algorithm can also be classified as an on-line or off-line algorithm according to when the characteristics of the whole set of tasks are available. If a task set must be known completely *a priori* in order to apply an algorithm, then the algorithm is referred to as being off-line, otherwise, it is said to be on-line.

Because real-time systems often operate in dynamic and complex environments, many scheduling decisions must be made on-line. For example, a change of mission may require the execution of a totally different task set. Or the failure of some processors may necessitate the re-assignment of tasks. In these scenarios, the entire task set to be scheduled may change dynamically, that is, tasks can be added or deleted from the task set. In the following, we present several on-line task assignment schemes for multiprocessor systems, where each processor executes the RM scheduling algorithm.

The solution to such a scheduling problem has practical implications. On the one hand, the real-time application domain is becoming increasingly large. As requirements of real-time support for industrial applications become more sophisticated, the employment of multiprocessors to meet computational power requirement seems inevitable. On the other hand, the state-of-the-art of hardware technology makes multiprocessor support a reality for many more systems. The scheduling of periodic tasks on a multiprocessor thus becomes an urgent problem that needs to be solved. From the perspective of RM scheduling, it has the property that as long as the CPU utilization of all tasks lies below a certain bound, all tasks will meet their deadlines without the programmer knowing exactly when any given instance of a task is running. Even if a transient overload occurs, a fixed subset of critical tasks will still meet their deadlines as long as their total CPU utilization lies within certain bound. The rate-monotonic scheduling has recently been used in a number of applications. For example, it has been specified for use with Space Station on-board software as the means for scheduling multiple independent task execution (Gafford, 1991). The RM algorithm will be built into the on-board operating system and is directly supported by the Ada compiler in use.

The problem of scheduling a set of periodic tasks on a multiprocessor systems has been addressed in a number of studies (Davari and Dhall, 1986a), (Davari and Dhall, 1986b), (Dhall and Liu, 1978). Typically, the task assignment schemes apply variants of well-known bin-packing heuristics where the set of processors is regarded as a set of bins. The bin-packing problem is concerned with packing different-sized items into fixed-sized bins using the least number of bins (Johnson, 1973), (Coffman et al., 1975). The decision whether a processor is full is determined by a schedulability condition. All existing task assignment schemes are based on the sufficient schedulability condition for uniprocessor systems derived by Liu and Layland (Liu and Layland, 1973) and its variants (Dhall and Liu, 1978).

In all studies, the performance of task assignment schemes is evaluated by providing the worst-case scenarios for N_A/N_0, where N_A is the number of processors required to

Table 1. Comparison of Task Assignment Schemes

	Existing Schemes				New Schemes	
Scheme A	RMNF	RMFF	NF-M	FFDUF	RRM-FF(BF)	RRM-FF(BF)
$\Re_A^\infty$	2.67	2.23?	[2.28,2.34]	2.0	2.33	2.0
Complexity	$O(n)$	$O(n \log n)$	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Type	*off-line*	*off-line*	*On-line*	*off-line*	*On-line*	*On-line*

schedule a task set by a given heuristic A, and N_0 is the number of processors needed by an optimal assignment. Worst-case bounds for the existing schemes are determined by the following expression: $\Re_A^\infty = \lim_{N_0 \to \infty} N_A/N_0$. In Table 1, we summarize the existing heuristic algorithms from the literature and our new schemes with their performance bounds. The measure $O(f(n))$ denotes the computation time complexity for scheduling a set of n real-time tasks.

Dhall and Liu were the first to propose two heuristic allocation algorithms for the scheduling problem (Dhall and Liu, 1978). The algorithms, *Rate-Monotonic-Next-Fit (RMNF)* and *Rate-Monotonic-First-Fit (RMFF)*, were shown to have worst-case performance bounds of $2.4 \leq \Re_{RMNF}^\infty \leq 2.67$, and $2 \leq \Re_{RMFF}^\infty \leq 4 \times 2^{1/3}/(1+2^{1/3}) \approx 2.23$. Unfortunately, the upper bound derived for RMFF was incorrect due to some errors in their proof. Furthermore, their RMFF is off-line, since it requires that tasks must be assigned in the order of increasing period.

Davari and Dhall later considered two other algorithms called *First-Fit-Decreasing-Utilization-Factor (FFDUF)* and *NEXT-FIT-M (NF-M)* (Davari and Dhall, 1986a), (Davari and Dhall, 1986b). The FFDUF algorithm sorts the set of tasks in non-increasing order of utilization factor and assigns tasks to processors in that order. The NEXT-FIT-M algorithm classified tasks into M classes with respect to their utilizations. Processors are also classified into M classes, so that a processor in k-class executes tasks in k-class exclusively. Their worst-case performance bounds are $\Re_{FFDUF}^\infty \leq 2$ and $\Re_{NF-M}^\infty \leq S_M$, respectively, where $S_M = 2.34$ for $M = 4$, and $S_M \to 2.2837$ when $M \to \infty$.

Since the decision whether a task can be assigned on a processor is determined by a schedulability condition, the nature and performance of an allocation algorithm is determined in part by the quality of the condition. The worst-case condition (or sufficient condition) of rate-monotonic scheduling has been presented by Liu and Layland in their seminal paper (Liu and Layland, 1973), and the necessary and sufficient condition was recently proven (Joseph and Pandya, 1986), (Lehoczky et al., 1989). While the famous condition given by $n(2^{1/n} - 1)$, where n is the number of tasks assigned to a processor, is too conservative in assigning tasks to processors, the necessary and sufficient condition is too complex to be used and analyzed in an allocation scheme.

Our approach to tackling the problem is to develop schedulability conditions that exhibit good performance while remaining simple enough so that the worst-case performance analysis is still possible. We then develop several simple allocation algorithms using the schedulability conditions. In the analysis of the worst-case performance, we not only obtain the upper bounds of the algorithms, but also provide examples that show the upper bounds are nearly tight. In the light of providing the worst-case performance, our analysis is non-trivial, because the algorithms are more complex than their bin-packing

counterparts, in the sense that the size of a bin is unitary in bin-packing, while the "size" or utilization of a processor is a variable. The value of the variable is determined by a function that is called the schedulability condition. We also derive the worst-case performance bounds of the algorithms with respect to the maximum allowable utilization of a task. Finally, the average-case performance of the algorithms are shown to be superior to those in the existing literature by simulation studies.

The rest of the paper is organized as follows: in the next section, the task model is described, and a new schedulability condition is presented. In Section 3, a new allocation algorithm called RM-FF is designed using the new schedulability condition. Its worst-case performance is analyzed. An improved version of RM-FF is given in Section 4. In Section 5, we present the Rate-Monotonic-Best-Fit (RM-BF) and its refined version, both of which are studied for their worst-case performance. In Section 6, we present our simulation methodology and the results of our experiments. We conclude in Section 7 and indicate our future research direction.

2. Task Model and A New Schedulability Condition

We assume that the tasks to be scheduled have the following characteristics:

1. The requests of each task are periodic, with constant interval between requests.
2. The deadline constraints specify that each request must be completed before the next request of the same task occurs.
3. The tasks are independent in the sense that the requests of a task do not depend on the initiation or the completion of the requests of other tasks.
4. Run-time (or computation time) for the request of a task is constant for the task. Run-time here refers to the time a processor takes to execute the request without interruption.

It follows that a task is completely defined by two numbers, the run-time of the requests and the request period. We shall denote a task τ_i by the ordered pair (C_i, T_i), where C_i and T_i are the computation time and the period of the requests, respectively. The ratio C_i/T_i, which is denoted as u_i, is called the utilization (or load) of the task τ_i. All the processors are identical processors in the sense that the run-time of a task remains the same across all processors. For the convenience of presentation, we adopt the following notations: processors are numbered in the order consistent with that of allocating them. P and Q are used to denote processors. $\tau_{x,l}$ denotes the lth task that is assigned on the xth processor; $u_{x,l}$ denotes the utilization of task $\tau_{x,l}$. τ_i is used to denote the ith task where there is no confusion. u_i denotes the utilization of the ith task on a processor or in a task set. U_j denotes the total CPU utilization (or load) of the jth processor. $\tau = (x, y)$ characterizes a task τ, where x and y are the computation time and the period of task τ, respectively.

The scheduling problem, which we call the *Rate-Monotonic Multiprocessor Scheduling (RMMS)* problem, can thus be described as follows:

Given a set of n tasks $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n\}$, where each task τ_i is characterized by its computation time C_i and its period T_i, what is the minimum number of processors required to execute the tasks such that their deadlines are met?

We say that a set of tasks is feasible if it can be scheduled by some algorithms such that all task deadlines are met. An optimal algorithm is one that uses the minimum number of processors for any given task set. The time to switch a processor from one task to another is negligible compared to the run-time of a task, and thus assumed to be zero.

If a set of periodic tasks can be feasibly scheduled on a single processor, then the Rate-Monotonic (RM) (Liu and Layland, 1973) or Intelligent Fixed Priority algorithm (Serlin, 1972) is optimal. The RM algorithm assigns priorities to tasks according to their periods, where the priority of a task is in inverse relationship to its period. In other words, a task with a shorter period is assigned a higher priority. The execution of a low-priority task will be preempted if a high-priority task arrives. The major result is stated in the following theorem.

THEOREM 1 *If a set of n tasks is scheduled according to the rate-monotonic algorithm, then the minimum achievable CPU utilization is $n(2^{1/n}-1)$. As $n \to \infty$, $n(2^{1/n}-1) \to \ln 2$ (Liu and Layland, 1973)(Serlin, 1972).*

This theorem ensures that a task set can be scheduled to meet their deadlines if the total utilization of the tasks is less than a threshold number, which is given by $n(2^{1/n} - 1)$, where n is the number of tasks in the task set. This condition—$\sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n} - 1)$, which is referred to as the WC condition, is a worst-case condition, since there are task sets which are feasible, but cannot be determined as feasible by the WC condition. For example, two tasks as given by $\tau_1 = (C_1, T_1) = (0.4, 1)$ and $\tau_2 = (C_2, T_2) = (0.5, 1)$ are infeasible according to the WC condition, since $\sum_{i=1}^{2} C_i/T_i > 2(2^{1/2}-1)$. But they are in fact feasible with the RM algorithm. There are task sets, however, that actually meet the worst-case condition. For example, a task set consists of $\tau_1 = (C_1, T_1) = (2^{1/2}-1, 1)$ and $\tau_2 = (C_2, T_2) = (2-2^{1/2}, 2^{1/2})$ with $\sum_{i=1}^{2} C_i/T_i = 2(2^{1/2}-1)$, where any increase in value of C_1 or C_2 will make the task set infeasible.

Another schedulability condition was given by Dhall and Liu (Dhall and Liu, 1978) while studying the performance of their multiprocessor heuristics—RMNF and RMFF.

THEOREM 2 *Let $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n\}$ be a set of n tasks with periods $T_1 \leq T_2 \leq \ldots \leq T_n$. Let $u = \sum_{i=1}^{n-1} C_i/T_i \leq (n-1)(2^{1/(n-1)} - 1)$. If $C_n/T_n \leq 2[1+u/(n-1)]^{-(n-1)} - 1$, then the set can be feasibly scheduled by the rate-monotonic algorithm. As $n \to \infty$, $2[1+u/(n-1)]^{-(n-1)} - 1 \to 2e^{-u} - 1$ (Dhall and Liu, 1978).*

This schedulability condition requires that the tasks be sorted in the order of non-decreasing period, thus implying that the task set should be known beforehand. Some of the task sets that cannot be scheduled by using the WC condition can be scheduled by using this condition, since this condition takes advantage of the fact that tasks are ordered against non-decreasing periods. This condition is referred to as the IP condition (*Increasing Period*).

Recently, the following result was obtained (Joseph and Pandya, 1986), (Lehoczky et al., 1989), which contains a condition that is both necessary and sufficient. This condition, referred to as the IFF condition, takes into account of both the computation time and period of a task.

THEOREM 3 *Let* $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n\}$ *be a set of* n *tasks.* τ_i *can be scheduled to meet its deadline using the rate-monotonic algorithm if and only if* $L_i = \min_{\{t \in S_i\}}(W_i(t)/t) \leq 1$. *The entire task set* Σ *is schedulable using the rate-monotonic algorithm if and only if* $L = \max_{\{1 \leq i \leq n\}} L_i \leq 1$, *where* $S_i = \{kT_j | j = 1, \ldots, i; k = 1, \ldots, \lfloor T_i/T_j \rfloor\}$, $W_i(t) = \sum_{j=1}^{i} C_j \lceil t/T_j \rceil$, $L_i = \min_{\{t \in S_i\}} L_i(t)$ *(Lehoczky et al., 1989).*

Note that the computation time requirement of the IFF condition is data dependent. In the worst cases, it may require more than exponential time complexity with respect to the number of tasks.

In the following we present a new schedulability condition, which is a sufficient condition, and yet is better in performance than any existing sufficient conditions.

THEOREM 4 *Let* $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n-1\}$ *be a set of* $(n-1)$ *tasks, and it can be feasibly scheduled by the rate-monotonic algorithm. Among the* $(n-1)$ *tasks, the utilizations of* $0 \leq m \leq n-1$ *tasks are known to be* $\{u_1, u_2, \ldots, u_m\}$, *and the total utilization of the rest of the* $(n-m-1)$ *tasks is known to be* u, *i.e.,* $u = \sum_{i=m+1}^{n-1} C_i/T_i$. *A new task* $\tau_n = (C_n, T_n)$ *can be feasibly scheduled with the* $(n-1)$ *tasks on a single processor by the rate-monotonic algorithm, if*

$$C_n/T_n \leq \begin{cases} 2[\prod_{i=1}^{m}(1+u_i)]^{-1} - 1 & \text{if } m = n-1 \\ 2[\prod_{i=1}^{m}(1+u_i)]^{-1}[1+u/(n-m-1)]^{-(n-m-1)} - 1 & \text{if } m < n-1 \end{cases} \quad (1)$$

This utilization condition is tight in the sense that there are task sets that actually meet this condition.

The introduction of m in the above schedulability conditions is to link the two extreme cases of $m = 0$ and $m = n-1$ together. The schedulability conditions apply as long as the utilizations of m tasks are known and the total utilization of the rest of $(n-m-1)$ tasks is also known. When $m = 0$, equation (1) becomes $C_n/T_n \leq 2[1+u/(n-1)]^{-(n-1)} - 1$. This condition is the same as the one given in Theorem 2 by Dhall and Liu, without the restriction that tasks must be assigned in the order of increasing period. The expression in (1) provides not only just one condition, but in an effect, $(n-1)$ conditions. Most significantly, no restriction is placed on the relative order of the tasks that are to be scheduled on a processor.

If we minimize the expression $U = \sum_{i=1}^{n} C_i/T_i$ over the condition given in expression (1), then the minimum is achieved when $u_i = 2^{1/n} - 1$ for $i = 1, 2, \ldots, n$, and $U = n(2^{1/n} - 1)$. This is exactly the condition given by Liu and Layland. In other words, the Liu and Layland's condition is a worst-case condition that is only achievable when each of the n tasks has the same utilization of $u_i = 2^{1/n} - 1$. If the utilizations of the tasks are

not the same, then the bound for $\sum_{i=1}^{n} C_i/T_i$ can be significantly higher than $n(2^{1/n}-1)$ in some cases under the new schedulability condition. For example, if $u_1 = 0.6$ and $u_2 = 0.1797$, then the maximum utilization of a task that can be scheduled together with the two tasks as given by expression (1) is $C_n/T_n \leq 2/[(1+u_1)(1+u_2)] - 1 = 0.06$, while it is impossible to schedule a third task on the same processor if the Liu and Layland's condition is used.

If we view the schedulability conditions for the rate-monotonic scheduling as points on a spectrum, then on the one end is the worst-case condition by Liu and Layland and on the other end is the sufficient and necessary condition by Joseph and Pandya (Joseph and Pandya, 1986), and Lehoczky et al. In the condition $\sum_{i=1}^{n} C_i/T_i \leq n(2^{1/n}-1)$, only the sum of the task utilizations is taken into account. In expression (1), starting from $m = 0$ to $m = n-1$, not only the sum of the task utilizations is taken into account, but also the utilization of each individual task. Furthermore, the condition can be expressed nicely in a formula, unlike the necessary and sufficient condition.

In order to prove Theorem 4, we need the following two lemmas.

LEMMA 1 *If a task (C_0, T_0) cannot be scheduled together with a set of $n \geq 1$ tasks $\{(C_1,T_1),(C_2,T_2),\ldots,(C_n,T_n)\}$ by the rate-monotonic algorithm and $T_0 \leq T_1 \leq \ldots \leq T_n$, then $(2C_0, 2T_0)$ cannot be scheduled together with the same set of tasks $\{(C_1,T_1),(C_2,T_2),\ldots,(C_n,T_n)\}$ by the rate-monotonic algorithm.*

Proof: Let us denote the task set of $\{(C_0,T_0),(C_1,T_1),\ldots,(C_n,T_n)\}$ with $T_0 \leq T_1 \leq \ldots \leq T_n$ as Σ, and the task set of $\{(2C_0,2T_0),(C_1,T_1),\ldots,(C_n,T_n)\}$ as Σ'. Suppose that for task set Σ, the ith task with $1 \leq i \leq n$ is the first task to miss its deadline. Then we claim that if $2T_0 \leq T_i$, then the ith task misses its deadline in the task set Σ', otherwise, task $(2C_0, 2T_0)$ misses its deadline.

Since (C_i, T_i) misses its deadline, according to the necessary and sufficient condition, we have

$f(t) = (C_0\lceil t/T_0\rceil + C_1\lceil t/T_1\rceil + \cdots + C_{i-1}\lceil t/T_{i-1}\rceil + C_i)/t > 1,$

for $t \in S = \{kT_j | j = 0,1,\ldots,i; k = 1,2,\ldots,\lfloor T_i/T_j\rfloor\} = \{T_0, 2T_0,\ldots,q_0T_0, T_1, 2T_1, \ldots, q_1T_1,\ldots,T_{i-1},\ldots,q_{i-1}T_{i-1}, T_i\}$, where $q_x = \lfloor T_i/T_x\rfloor$ for $x = 0,2,\ldots,i-1$.

Case 1: $2T_0 \leq T_i$. Let us examine the new function:

$f'(t) = \lceil 2C_0\lceil t/(2T_0)\rceil + C_1\lceil t/T_1\rceil + \cdots + C_{i-1}\lceil t/T_{i-1}\rceil + C_i\rceil/t,$

for $t \in S' = \{kT_j | j = 0',1,2,\ldots,i; k = 1,2,\ldots,\lfloor T_i/T_j\rfloor\} = \{2T_0, 4T_0,\ldots,2q_{0'}T_0, T_1, 2T_1,\ldots,q_1T_1,\ldots,T_{i-1},2T_{i-1},\ldots,q_{i-1}T_{i-1},T_i\}$, where $C_{0'} = 2C_0$ and $T_{0'} = 2T_0$, and $q_{0'} = \lfloor T_i/(2T_0)\rfloor$.

Since $C_0\lceil t/T\rceil = 2C_0\lceil t/(2T_0)\rceil$ for $t \in \{2T_0, 4T_0,\ldots,2q_{0'}T_0\}$, we have $f'(t) = f(t) > 1$.

For $t \in \{T_1, 2T_1,\ldots,q_1T_1,\ldots,T_{i-1},2T_{i-1},\ldots,q_{i-1}T_{i-1},T_i\}$, we claim that $2C_0\lceil t/(2T)\rceil \geq C_0\lceil t/T\rceil$.

Since $t > T_0$, we write $t = wT_0 + r$, where $0 \leq r < T_0$ and $w \geq 1$. If $r = 0$, then $2\lceil t/(2T_0)\rceil = 2\lceil w/2\rceil \geq w = \lceil t/T_0\rceil$. If $r > 0$, then $2\lceil t/(2T_0)\rceil = 2\lceil q/2 + r/(2T_0)\rceil \geq \lceil q + r/T_0\rceil$. Therefore, $f'(t) \geq f(t) > 1$. In other words, task (C_i, T_i) misses its deadline.

Case 2: $T_i < 2T_0$. Let us examine the new function:

$f'(t) = (C_1\lceil t/T_1\rceil + C_2\lceil t/T_2\rceil + \cdots + C_{i-1}\lceil t/T_{i-1}\rceil + C_i\lceil t/T_i\rceil + 2C_0)/t,$
for $t \in S' = \{T_1, T_2, \ldots, T_{i-1}, T_i, 2T_0\}$.

Since $\lceil T_j/T_0\rceil = 2$ and $\lceil T_j/T_i\rceil = 1$ for $j = 1, 2, \ldots, i$, we have $f'(t) = f(t) > 1$, for $t \in \{T_1, T_2, \ldots, T_{i-1}, T_i\}$. For $t = 2T_0$, $f'(t) = (2C_1 + 2C_2 + \cdots + 2C_i + 2C_0)/(2T_0)$ $= (C_1 + C_2 + \cdots + C_i + C_0)/T_0 = f(T_0) > 1$. In other words, task $(2C_0, 2T_0)$ misses its deadline.

Therefore, the lemma is true. ■

Lemma 1 is a very powerful lemma, since it implies that if one task set is infeasible, then there exists another that is also infeasible with the same task utilizations, but with the ratio between any two task periods less than 2. Therefore, in deriving schedulability condition, we need only to consider task sets where the ratio between any two task periods is less than 2.

The following lemma is a reiteration of a fact that was used implicitly by Liu and Layland to derive their worst-case condition.

LEMMA 2 *For a set of n tasks* $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n\}$ *with fixed priority order, and the restriction that* $T_1 \le T_2 \le \ldots \le T_n < 2T_1$, *the least upper bound to the processor utilization is achieved when* $C_i = T_{i+1} - T_i$ *for* $i = 1, 2, \ldots, n-1$ *and* $C_n = 2T_1 - T_n$.

Proof: Let $\Sigma = \{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n\}$ be a set of n tasks with $T_1 \le T_2 \le \ldots \le T_n < 2T_1$, and $C_1, C_2, \ldots, C_n$ be the computation times of the tasks that fully utilize the processor and minimize the processor utilization, i.e., $U = \sum_{i=1}^{n} C_i/T_i$.

Suppose that

$$C_1 = T_2 - T_1 + \Delta, \Delta > 0.$$

Let

$$\begin{aligned} C_1' &= T_2 - T_1, \\ C_2' &= C_2 + \Delta, \\ C_3' &= C_3, \\ &\cdots \\ C_n' &= C_n. \end{aligned}$$

Then $C_1', C_2', \ldots, C_n'$ also fully utilize the processor. Let $U' = \sum_{i=1}^{n} C_i'/T_i$. Then

$$U - U' = (\Delta/T_1) - (\Delta/T_2) > 0.$$

On the other hand, suppose that

$$C_1 = T_2 - T_1 - \Delta, \Delta > 0.$$

Let

$$C_1' = T_2 - T_1,$$
$$C_2' = C_2 - 2\Delta,$$
$$C_3' = C_3,$$
$$\ldots$$
$$C_n' = C_n.$$

Then again $C_1', C_2', \ldots, C_n'$ also fully utilize the processor. Let $U' = \sum_{i=1}^{n} C_i'/T_i$. Then

$U - U' = -(\Delta/T_1) + (2\Delta/T_2) > 0.$

Therefore, if U is indeed the minimum processor utilization, then

$C_1 = T_2 - T_1.$

Similarly we can show that

$C_i = T_{i+1} - T_i$, for $i = 2, \ldots, n-1$, and

$C_n = T_1 - \sum_{i=1}^{n-1} C_i = 2T_1 - T_n.$

Thus the lemma is proven. ■

Proof of Theorem 4: We will obtain the expression in (1) through two steps: one is under the condition $m < n - 1$, and the other is under the condition $m = n - 1$.

Let us first form a new task set called Σ, from the task τ_n and the set of $(n-1)$ tasks $\{\tau_i = (C_i, T_i) | i = 1, 2, \ldots, n-1\}$, i.e., $\Sigma = \{\tau_i | i = 1, 2, \ldots, n\}$.

According to Lemma 1, we can assume, without loss of generality, that the periods of the task set satisfy

$T_i/T_j < 2$, for $i, j = 1, 2, \ldots, n$ and $i \neq j$.

We then sort the tasks in the order of the increasing period and rename them appropriately such that

$T_1 \leq T_2 \leq \ldots \leq T_n < 2T_1.$

Then according to Lemma 2, the minimum utilization is achieved when

$$C_1 = T_2 - T_1,$$

$$C_2 = T_3 - T_2,$$

$$\ldots$$

$$C_{n-1} = T_n - T_{n-1},$$

$$C_n = T_1 - \sum_{i=1}^{n-1} C_i = 2T_1 - T_n.$$

This set of tasks fully utilizes the processor, even though there is an unknown quantity—C_i/T_i in the above conditions, that has yet to be determined. The unknown quantity

C_i/T_i corresponds to the original C_n/T_n before renaming. Let $U = \sum_{i=1}^{n} C_i/T_i$ denote the total utilization of the new task set. Now pbserve that the above conditions are symmetric in the sense that if we multiply 2 to the computation time and period of the first task τ_1, then the utilization of each task (and hence the total utilization of the n tasks) remains unchanged, and the new task set still fully utilizes the processor. Therefore, by applying the multiplying rule in no more than n steps, we can arrive at a new task set where τ_n is the largest period among all n tasks, with C_n/T_n as the unknown quantity. Furthermore, we can let $u_i = C_i/T_i$ for $i = 1, 2, \ldots, m$ and $u = \sum_{i=m+1}^{n-1} C_i/T_i$.

Case 1: $m = n - 1$. Then

$$U = \sum_{i=1}^{n} C_i/T_i = \sum_{i=1}^{m} u_i + C_n/T_n.$$

where $u_i = C_i/T_i = (T_{i+1} - T_i)/T_i = T_{i+1}/T_i - 1$ for $i = 1, 2, \ldots, n-1$.

Let $x_i = T_{i+1}/T_i$ for $i = 1, 2, \ldots, n-1$. Then $C_n/T_n = (2T_1 - T_n)/T_n = 2/\prod_{i=1}^{n-1} x_i - 1$. But since $x_i = u_i + 1$,

$$C_n/T_n = 2/\prod_{i=1}^{n-1} x_i - 1 = 2/\prod_{i=1}^{m}(1+u_i) - 1.$$

The task sets that actually meet this condition are given as follows:

Let $T_1 = \sigma > 0$, then $C_1 = \sigma u_1$. Hence we have $T_{i+1} = (1+u_i)T_i = \sigma \prod_{j=1}^{i}(1+u_j)$, $C_{i+1} = u_{i+1}T_{i+1} = \sigma u_{i+1} \prod_{j=1}^{i}(1+u_j)$, for $i = 1, 2, \ldots, n-2$, and $T_n = \sigma \prod_{j=1}^{n-1}(1+u_j)$, $C_n = \sigma[2 - \prod_{j=1}^{n-1}(1+u_j)]$. In other words, the task sets are given as

$$(C_1, T_1) = (\sigma u_1, \sigma),$$

$$(C_2, T_2) = (\sigma u_2(1+u_1), \sigma(1+u_1)),$$

$$\ldots$$

$$(C_{n-1}, T_{n-1}) = (\sigma u_{n-1} \prod_{j=1}^{n-2}(1+u_j), \sigma \prod_{j=1}^{n-2}(1+u_j)),$$

$$(C_n, T_n) = (\sigma(2 - \prod_{j=1}^{n-1}(1+u_j)), \sigma \prod_{j=1}^{n-1}(1+u_j)).$$

Case 2: $m < n - 1$. The utilization of task is determined at the point where the total utilization is minimized.

$$U = \sum_{i=1}^{n} C_i/T_i = \sum_{i=1}^{m} u_i + u + C_n/T_n,$$

where $u_i = C_i/T_i = (T_{i+1} - T_i)/T_i = T_{i+1}/T_i - 1$ for $i = 1, 2, \ldots, m$ and $u = \sum_{i=m+1}^{n-1} C_i/T_i$.

Let $x_i = T_{i+1}/T_i$ for $i = 1, 2, \ldots, n-1$. Then $C_n/T_n = (2T_1 - T_n)/T_n = 2/\prod_{i=1}^{n-1} x_i - 1$. After proper substitution, we obtain

$$U = \sum_{i=1}^{m} u_i + u + 2/\prod_{i=1}^{n-1} x_i - 1, \tag{2}$$

$$u_i = x_i - 1, for\ i = 1, 2, \ldots, m, \tag{3}$$

$$u = \sum_{i=m+1}^{n-1} C_i/T_i = \sum_{i=m+1}^{n-1} x_i - (n - m - 1), \tag{4}$$

To find the minimum, we need to minimize the expression for U as given in equation (2) subject to the side conditions (3) and (4). This is achieved by first forming the Lagrangian

$$L = U + \sum_{i=1}^{m} \lambda_i (x_i - 1 - u_i) + \lambda [\sum_{i=m+1}^{n-1} x_i - (n - m - 1) - u],$$

and then minimizing the function L over x_i's, λ_i's, and λ. This can be accomplished by first taking the derivatives of L over x_i's, λ_i's, and λ, respectively, and then solving the resultant equations after setting them to zero.

$$\frac{\partial L}{\partial x_i} = \lambda_i - \frac{2}{x_i \prod_{j=1}^{n-1} x_j} = 0, for\ i = 1, 2, \ldots, m. \tag{5}$$

$$\frac{\partial L}{\partial x_i} = \lambda - \frac{2}{x_i \prod_{j=1}^{n-1} x_j} = 0, for\ i = m+1, m+2, \ldots, n-1. \tag{6}$$

$$\frac{\partial L}{\partial \lambda_i} = x_i - 1 - u_i = 0, for\ i = 1, 2, \ldots, m. \tag{7}$$

$$\frac{\partial L}{\partial \lambda} = \sum_{i=m+1}^{n-1} x_i - (n - m - 1) - u = 0. \tag{8}$$

In the following we show how the above equations can be solved to obtain the final results.

By multiplying the $(n-1)$ equations in (5) and (6) together and manipulating the resultant equation, we get

$$\prod_{j=1}^{n-1} x_j = \frac{2^{(n-1)/n}}{\lambda^{(n-m-1)/n}(\prod_{i=1}^{m}\lambda_i)^{1/n}}. \tag{9}$$

By substituting the $\prod_{j=1}^{n-1} x_j$ in (5) and (6) by that in (9) and solving for x_i's, we have

$$x_i = \frac{2^{1/n}\lambda^{(n-m-1)/n}(\prod_{i=1}^{m}\lambda_i)^{1/n}}{\lambda_i}, \text{for } i = 1, 2, \ldots, m. \tag{10}$$

$$x_i = \frac{2^{1/n}(\prod_{i=1}^{m}\lambda_i)^{1/n}}{\lambda^{(m+1)/n}}, \text{for } i = m+1, m+2, \ldots, n-1. \tag{11}$$

Since $x_i = 1 + u_i$, we have from (10) that

$$1 + u_i = \frac{2^{1/n}\lambda^{(n-m-1)/n}(\prod_{i=1}^{m}\lambda_i)^{1/n}}{\lambda_i}, \text{for } i = 1, 2, \ldots, m. \tag{12}$$

By multiplying the m equations in (12) together and manipulating the resultant equation, we get

$$\prod_{i=1}^{m}\lambda_i = 2^{m/(n-m)}\lambda^{m(n-m-1)/(n-m)}[\prod_{i=1}^{m}(1+u_i)]^{n/(m-n)}. \tag{13}$$

Since $\sum_{i=m+1}^{n-1} x_i = (n-m-1) + u$ from (4) and $\sum_{i=m+1}^{n-1} x_i = (n-m-1)\frac{2^{1/n}(\prod_{i=1}^{m}\lambda_i)^{1/n}}{\lambda^{(m+1)/n}}$ from (11), we have

$$\prod_{i=1}^{m}\lambda_i = \frac{\lambda^{m+1}}{2}[1 + u/(n-m-1)]^n. \tag{14}$$

For convenience, we let $\Delta = 1 + \frac{u}{n-m-1}$ and $\nabla = \prod_{i=1}^{m}(1+u_i)$. With equations (13) and (14) together, we can solve for λ:

$$\lambda = \frac{2}{\nabla\Delta^{n-m}}. \tag{15}$$

With equations (12), (14), and (15) together we can solve for λ_i's:

$$\lambda_i = \frac{2}{(1+u_i)\nabla\Delta^{n-m-1}}, \text{for } i = 1, 2, \ldots, m.$$

Then we have

$$\prod_{i=1}^{m}\lambda_i = \frac{2^m}{\nabla^{m+1}\Delta^{(n-m-1)m}}. \tag{16}$$

Now we are ready to solve for x_i's. With equations (11), (15), and (16) together we obtain

$$x_i = \frac{2^{1/n}(\prod_{i=1}^{m}\lambda_i)^{1/n}}{\lambda^{(m+1)/n}} = \Delta, \text{for } i = m+1, m+2, \ldots, n-1. \tag{17}$$

Since $x_i = 1 + u_i$ for $i = 1, 2, \ldots, m$, we have

$$\frac{C_n}{T_n} = \frac{2}{\prod_{i=1}^{n-1} x_i} - 1 = \frac{2}{\nabla\Delta^{n-m-1}} - 1.$$

In other words, if

$$\frac{C_n}{T_n} \leq \frac{2}{\prod_{i=1}^{m}(1+u_i)[1+u/(n-m-1)]^{n-m-1}} - 1,$$

then the new task set can be feasibly scheduled by the Rate-Monotonic algorithm. As $n \to \infty$, we have

$$\frac{2}{(\nabla\Delta^{n-m-1})} - 1 \longrightarrow 2e^{-u}/\prod_{i=1}^{m}(1+u_i) - 1.$$

The task sets that actually meet this condition are given as follows: Let $T_1 = \sigma > 0$, then $C_1 = \sigma u_1$. Hence we have $T_{i+1} = (1+u_i)T_i = \sigma\prod_{j=1}^{i}(1+u_j)$, $C_{i+1} = u_{i+1}T_{i+1} = \sigma u_{i+1}\prod_{j=1}^{i}(1+u_j)$, for $i = 1, 2, \ldots, m-1$, $T_{m+1} = x_m T_m = \sigma\prod_{j=1}^{m}(1+u_j)$, $T_{i+1} = x_i T_i = \Delta T_i = \Delta^{i-m}\sigma\prod_{j=1}^{m}(1+u_j)$, $C_i = T_{i+1} - T_i = (\Delta-1)\Delta^{i-m}\sigma\prod_{j=1}^{m}(1+u_j)$, for $i = m+1, m+2, \ldots, n-2$, $C_n = \sigma[2-\prod_{j=1}^{n-1}(1+u_j)]$. In other words, with σ as a variable, the task sets are given as

$$(C_1, T_1) = (\sigma u_1, \sigma),$$

$$(C_2, T_2) = (\sigma u_2(1+u_1), \sigma(1+u_1)),$$

$$\ldots$$

$$(C_{m-1}, T_{m-1}) = (\sigma u_{m-1}\prod_{j=1}^{\cdot\cdot}(1 \cdot u_j), \sigma\prod_{j=1}^{m-2}(1+u_j)),$$

$$(C_m, T_m) = (\sigma u_m\prod_{j=1}^{m-1}(1+u_j), \sigma\prod_{j=1}^{m-1}(1+u_j)),$$

$$(C_{m+1}, T_{m+1}) = (\sigma(\Delta-1)\prod_{j=1}^{m}(1+u_j), \sigma\prod_{j=1}^{m}(1+u_j)),$$

...

$$(C_{n-2}, T_{n-2}) = (\sigma(\triangle - 1)\triangle^{n-3-m} \prod_{j=1}^{m}(1+u_j), \sigma\triangle^{n-3-m} \prod_{j=1}^{m}(1+u_j)),$$

$$(C_{n-1}, T_{n-1}) = (\sigma(\nabla\triangle^{n-m-1} - \triangle^{n-m-2}), \sigma\triangle^{n-2-m} \prod_{j=1}^{m}(1+u_j)),$$

$$(C_n, T_n) = (\sigma(2 - \nabla\triangle^{n-m-1}), \sigma\triangle^{n-m-1}\nabla).$$

3. The Design and Analysis of Rate-Monotonic-First-Fit

The problem of assigning tasks onto the minimum number of processors very closely resembles the bin-packing problem, in which items of variable sizes are packed into as few bins as possible. Therefore, many of the bin-packing heuristics can be adapted to assign tasks to processors. However, the key difference between the bin-packing problem and the RMMS problem is that the "size" or utilization of a processor is not always unitary, but rather it is a variable whose values are determined by the schedulability condition. Among the many bin-packing heuristics, First-Fit has the nice property of being on-line, simple, and efficient. Its worst-case performance is near to the that of an optimal algorithm (Coffman et al., 1975) (First-Fit has a worst-case tight bound of 1.7, while any on-line bin-packing heuristic must have a worst-case bound of at least 1.536. See (Coffman et al., 1975) for details). We will use the First-Fit strategy to assign tasks to processors in our new scheduling algorithm, called Rate-Monotonic-First-Fit (RM-FF). Note that the name of the new algorithm is only one "dash" different from Dhall and Liu's RMFF.

The RM-FF algorithm is designed as follows: let the processors be indexed as $P_1, P_2, \ldots$, with each one initially in the idle state, i.e., with zero utilization. The tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ will be scheduled in that order. To schedule τ_i, find the least j such that task τ_i, together with all the tasks that have been assigned to processor P_j, can be feasibly scheduled according to the new schedulability condition for a single processor, and assign task τ_i to P_j.

To describe RM-FF in a more algorithmic format, we let k_j and u_j denote the number of tasks that have already been assigned to processor P_j so far and the total utilization of the k_j tasks, respectively. Note that u_i denotes the utilization of task τ_i, and $u_{i,j}$ denotes the utilization of the ith task assigned on processor P_j.

Rate-Monotonic-First-Fit (RM-FF) (Input: task set Σ; Output: m)

(1) $i := 1$; $m := 1$;

(2) $j := 1$; **While** $(u_i > \frac{2}{\prod_{l=1}^{k_j}(1+u_{j,l})} - 1)$ **Do** $j := j + 1$;
(3) $k_j := k_j + 1$; $U_j := U_j + u_i$; // Assign task τ_i to P_j
(4) **If** $(j > m)$ **Then** $m = j$;
(5) $i := i + 1$;
(6) **If** $(i > n)$ **Then Exit Else Goto** 2;

When the algorithm terminates, m is the number of processors required by RM-FF to schedule the given set of tasks. The RM-FF algorithm has the following distinguished property: no incoming task is assigned to an idle processor unless it cannot be assigned to any processor that has already been assigned some tasks. We will implicitly use this property throughout the analysis. In order to obtain the worst-case bound, we need some lemmas.

LEMMA 3 *If n tasks cannot be feasibly scheduled on $n-1$ processors according to RM-FF, then the total utilization of the n tasks is greater than $n/(1+2^{1/2}) = n(2^{1/2}-1)$.*

Proof: The proof is by induction.

(1) $n = 2$. Suppose u_1 and u_2 are the utilizations of two tasks which cannot be scheduled on a processor according to the new condition, i.e., $u_2 > 2(1+u_1)^{-1} - 1$. Then $u_1 + u_2 > u_1 + 2(1+u_1)^{-1} - 1$. To find the minimum of $f(u_1) = u_1 + 2(1+u_1)^{-1} - 1$, we take the derivative of the function $f(u_1)$ and solve for u_1. The minimum of $f(u_1)$ is achieved when $u_1 = (2^{1/2} - 1)$. Therefore $u_1 + u_2 > 2(2^{1/2} - 1)$.

(2) Suppose the lemma is true for $n = k$, i.e.,

$$\sum_{i=1}^{k} u_i > k(2^{1/2} - 1), \tag{18}$$

where u_i is the utilization of task τ_i.

When $n = k + 1$, the $(k+1)th$ task cannot be scheduled on any of the k processors, i.e., $u_i + u_{k+1} > 2(2^{1/2} - 1)$, where $1 \le i \le k$. Summing up the k equations yields

$$\sum_{i=1}^{k} u_i + k u_{k+1} > 2k(2^{1/2} - 1) \tag{19}$$

Multiplying $(k-1)$ on both sides of equation (18) yields

$$(k-1)\sum_{i=1}^{k} u_i > (k-1)k(2^{1/2} - 1) \tag{20}$$

Adding up equations (19) and (20) and dividing the new equation on both sides by k yields $\sum_{i=1}^{k+1} u_i > (k+1)(2^{1/2} - 1)$. Therefore Lemma 3 is proven. ■

LEMMA 4 *In the final RM-FF schedule, among all the processors on which $n \geq c \geq 1$ tasks are assigned, there is at most one processor with a utilization less than or equal to $c(2^{1/(c+1)} - 1)$.*

Proof: The lemma is proven by contradiction. Suppose there are two processors each of which has a utilization less than or equal to $c(2^{1/(c+1)} - 1)$, and let P_1 and P_2 be the two processors, and n_i be the number of tasks assigned to processor P_i with $n_i \geq c$ and $i = 1, 2$. Let $u_{i,j}$ be the utilization of the jth task that is assigned to processor P_i, for $i = 1, 2$ and $1 \leq j \leq n_i$. Then $\sum_{j=1}^{n_i} u_{i,j} \leq c(2^{1/(c+1)} - 1)$ for $i = 1, 2$.

If there exists a task $\tau_{2,x}$ such that $u_{2,x} \geq (2^{1/(c+1)} - 1)$ for $1 \leq x \leq n_2$, then there exists a task with a utilization $u_{2,y} \leq (2^{1/(c+1)} - 1)$, since there are totally $n_2 \geq c$ tasks on processor P_2 and $\sum_{j=1}^{n_2} u_{2,j} \leq c(2^{1/(c+1)} - 1)$, where $x \neq y$ and $1 \leq y \leq n_2$. In other words, there exists a task $\tau_{2,z}$ on processor P_2 satisfying $u_{2,z} \leq (2^{1/(c+1)} - 1)$, where $z \in \{1, 2, \ldots, n_2\}$.

Since $\sum_{j=1}^{n_1} u_{1,j} + u_{2,z} \leq c(2^{1/(c+1)} - 1) + (2^{1/(c+1)} - 1) = (c+1)(2^{1/(c+1)} - 1)$, $\tau_{2,z}$ can be assigned on processor P_1. This is a contradiction to the way RM-FF assigns tasks to processors. Therefore the lemma must be true. ■

THEOREM 5 *Let N and N_0 be the number of processors required by RM-FF and the minimum number of processors required to feasibly schedule a given set of tasks, respectively. Then $N \leq [2 + (3 - 2^{3/2})/(2^{4/3} - 2)]N_0 + 1 \approx 2.33N_0 + 1$.*

In order to prove the above bound, we define a weighting function that maps the utilization of a task into the real interval $(0, 1]$ as follows:

$$f(u) = \begin{cases} u/a & 0 < u < a \\ 1 & a \leq u \leq 1 \end{cases},$$

where $a = 2(2^{1/3} - 1)$.

LEMMA 5 *If a processor is assigned a number of tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ with utilizations $u_1 \geq u_2 \geq \ldots \geq u_n$, then $\sum_{i=1}^{n} f(u_i) \leq 1/a$, where $a = 2(2^{1/2} - 1)$.*

Proof: If $u_1 \geq a$, then $u_2 < a$, since $a \approx 0.52$. $\sum_{i=1}^{n} f(u_i) = f(u_1) + \sum_{i=2}^{n} f(u_i) = 1 + \sum_{i=2}^{n} u_i/a \leq 1 + (1-a)/a = 1/a$. Otherwise ($u_1 < a$), $\sum_{i=1}^{n} f(u_i) = \sum_{i=1}^{n} u_i/a \leq 1/a$. ■

LEMMA 6 *Suppose tasks are assigned to processors according to RM-FF. If a processor is assigned $n \geq 2$ tasks and $\sum_{i=1}^{n} u_i \geq 2(2^{1/3} - 1)$, then $\sum_{i=1}^{n} f(u_i) \geq 1$, where $u_1 \geq u_2 \geq \ldots \geq u_n$ are utilizations of the n tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ that are assigned to it.*

Proof: Since $\sum_{i=1}^{n} u_i \geq 2(2^{1/3} - 1)$, we either have $u_1 \geq a$ or $u_1 < a$. If $u_1 \geq a$, then $\sum_{i=1}^{n} f(u_i) \geq 1$ according to the definition of weighting function. Otherwise, $\sum_{i=1}^{n} f(u_i) = \sum_{i=1}^{n} u_i/a > 1$, where $a = 2(2^{1/3} - 1)$. ■

Proof of Theorem 5: Let $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n tasks with their utilizations $u_1 \geq u_2 \geq \ldots \geq u_n$, respectively, and $\omega = \sum_{i=1}^{n} f(u_i)$. By Lemma 5, $\omega \leq N_0/a$, where $a = 2(2^{1/3} - 1)$.

Suppose that among the N processors that are used by RM-FF to schedule a given set Σ of tasks, L of them has $\sum_j f(u_j) = 1 - \beta_i$ with $\beta_i > 0$, where j ranges over all tasks in processor P_i among the L processors. Let us divide these processors into two different classes:

(1) Processors to each of which only one task is assigned. Suppose there are n_1 of them.

(2) Processors to each of which two or more tasks are assigned. Let n_2 denote the number of processors in this class. According to Lemma 4, there is at most one processor whose utilization in the RM-FF schedule is less than or equal to $a = 2(2^{1/3} - 1)$. Therefore $n_2 \leq 1$.

Obviously, $L = n_1 + n_2$. For each of the rest $N - L$ processors, $\sum_j f(u_j) \geq 1$, where j ranges over all tasks in a processor.

For the processors in class (1), $\sum_{i=1}^{n_1} u_i > n_1(2^{1/2} - 1)$ according to Lemma 3. Since $\sum_{i=1}^{n_1} f(u_i) < 1$, it holds that $u_i < a$. Therefore, $\sum_{i=1}^{n_1} f(u_i) > n_1(2^{1/2} - 1)/a$. Moreover, according to Lemma 4, there is at most one task whose utilization is less than or equal to $(2^{1/2} - 1)$. In the optimal assignment of these tasks, the optimal number N_0 of processors used cannot be less than $n_1/2$, i.e., $N_0 \geq n_1/2$, since possibly with one exception, any three tasks among these tasks cannot be scheduled on one processor.

Now we are ready to find out the relationship between N and N_0.

$$\begin{aligned}\sum_{i=1}^{n} f(u_i) &\geq (N - L) + n_1(2^{1/2} - 1)/a = N - n_1 - n_2 + n_1(2^{1/2} - 1)/a \\ &= N - n_1[1 - (2^{1/2} - 1)/a] - n_2 \\ &\geq N - 2N_0[1 - (2^{1/2} - 1)/a] - n_2.\end{aligned}$$

Since $\sum_{i=1}^{n} f(u_i) = \omega \leq N_0/a$ by Lemma 5,

$$N_0/a \geq N - 2N_0(1 - (2^{1/2} - 1)/a) - n_2 \geq N - 2N_0[1 - (2^{1/2} - 1)/a] - 1.$$

Therefore, $N \leq [2 + (3 - 2^{3/2})/a]N_0 + 1$. ■

Having proven the upper bound for RM-FF, we construct a number of task sets, each of which, when scheduled by RM-FF, requires nearly the same upper bounded number of processors. This theorem also serves as an counter example for the claim that RMFF is upper bounded by 2.23 in (Dhall and Liu, 1978). Since the tasks in each group has the same utilization, and to show the incorrectness of the upper bound for RMFF in (Dhall and Liu, 1978), we use the condition—$C_n/T_n \leq 2[1 + u/(n-1)]^{-(n-1)} - 1$ in the place of the condition—$C_n/T_n \leq 2/\prod_{i=1}^{n-1}(1 + u_i) - 1$. No difference is resulted in with this replacement.

THEOREM 6 *Let N and N_0 be the number of processors required by RM-FF and the minimum number of processors required to feasibly schedule a given set of tasks, respectively. Then*

$$\Re^{\infty}_{RM-FF} = \lim_{N_0 \to \infty} N/N_0 \geq 2.2833\ldots$$

Proof: The proof consists of constructing task sets such that RM-FF exhibits the worst-case performance when it is used to schedule them.

Let $n = 120k$ with $k \geq 1$. The task set is given as follows: first n tasks are given, each of which has a utilization of $u_1 = 2^{1/31} - 1 + \epsilon$, where ϵ is a small positive number. For our construction, it suffices to let $\epsilon < 0.00006$. Next come n tasks, each with a utilization of $u_2 = 2^{1/5} - 1 + \epsilon$. Finally, there are $2n$ tasks, each with a utilization of $u_3 = 2^{1/2} - 1 + \epsilon$.

When this task set is scheduled by RM-FF, the first n tasks will use $\lfloor n/30 \rfloor$ processors, since $2^{1/31} - 1 + \epsilon > 2\{1 + [30(2^{1/31} - 1 + \epsilon)]/30\}^{-30} - 1$. The next n tasks will take up $\lfloor n/4 \rfloor$ processors, since $2^{1/5} - 1 + \epsilon > 2\{1 + [4(2^{1/5} - 1 + \epsilon)]/4\}^{-4} - 1$. The last $2n$ tasks will take up $2n$ processors for similar reason. Thus, the total number of processors allocated for this task set by RM-FF is $N = 2n + \lfloor n/4 \rfloor + \lfloor n/30 \rfloor = 274k$.

For the optimal schedule, each processor is assigned four tasks and a total of n processors is required. More specifically, for each processor, it is assigned two tasks each with a utilization of $(2^{1/2} - 1 + \epsilon)$, one task of $(2^{1/5} - 1 + \epsilon)$, and one task of $(2^{1/31} - 1 + \epsilon)$, since $2(2^{1/2} + \epsilon) + (2^{1/5} - 1 + \epsilon) + (2^{1/31} - 1 + \epsilon) < 1$ for $\epsilon < 0.00006$. Therefore, $N_0 = n = 120k$. The schedule by RM-FF and the optimal schedule are given in Figure 1.

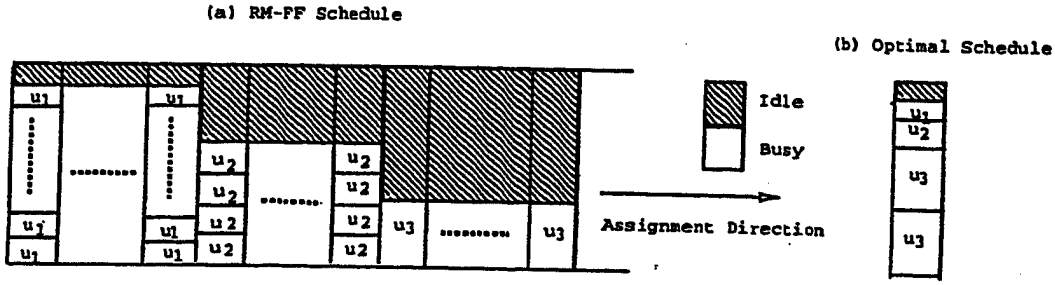


Figure 1. RM-FF vs. Optimal Schedule.

Then the asymptotic bound of RM-FF satisfies

$$\Re^{\infty}_{RM-FF} = \lim_{N_0 \to \infty} N/N_0 = 274k/(120k) \geq 2.2833\ldots$$

■

Theorem 6 shows that the number of processors required by RM-FF to schedule some task sets can be 2.283 times that of the optimal number of processors. Since Theorem 5

Table 2. The Worst-Case Performance Bounds of RM-FF under α

α	> 0.4142	< 0.4142	< 0.2599	< 0.1892	< 0.1487	< 0.02
$\Re_A^\infty$	2.33	1.92	1.76	1.68	1.63	1.47

ensures that the number of processors required by RM-FF to schedule any given task set cannot exceed 2.33 times that of the optimal number of processors, we conclude that the bound of 2.33 is almost tight. Note that the lower bound of 2.283 may be improved or pushed up since the above construction does not guarantee that the bound of 2.283 is indeed the worst-case lower bound.

4. The Design and Analysis of Refined-Rate-Monotonic-First-Fit

The bounds for RM-FF were derived under the general assumption that the utilization of a task can take any value between zero and one. If the utilization of a task is small compared to the processing power of a processor—a very realistic assumption considering the state-of-the-art hardware technology—we show that the worst-case performance of RM-FF can be significantly improved.

Let α be the maximum allowable utilization of a task, i.e., $\alpha = \max_i\{C_i/T_i\}$. Then we have the following theorem.

THEOREM 7 *Let N and N_0 be the number of processors required by RM-FF and the minimum number of processors required to feasibly schedule a given set of tasks, respectively. Further let $\alpha = \max_{1\le i\le n}\{u_i\}$. If $\alpha \le 2^{1/(1+c)} - 1$, then*

$$\Re^\infty_{RM-FF}(\alpha) \le \frac{1}{(c+1)(2^{1/(2+c)}-1)}, \text{for } c = 0,1,2,\ldots$$

When $c \to \infty$, $(c+1)(2^{1/(2+c)}-1) \to \ln 2$. Then $\Re^\infty_{RM-FF} \le 1/\ln 2$. The upper bounds of RM-FF with regard to α are given in Table 2 for a few values of α.

Proof: For any set of n tasks, let $\sum_{i=1}^n u_i$ be the total utilization of the task set. According to RM- FF, if $\alpha \le 2^{1/(1+c)} - 1$, then each processor must be assigned at least $(c+1)$ tasks since $\alpha(c+1) \le (c+1)(2^{1/(1+c)}-1)$, except possibly for the last processor. According to Lemma 4, among all the processors to each of which at least $(c+1)$ tasks are assigned, there is at most one processor whose utilization is less than or equal to $(c+1)(2^{1/(2+c)}-1)$, for $c = 0,1,2,\ldots$ Since $(N-2)(c+1)[2^{1/(2+c)}-1] \le \sum_{i=1}^n \le N_0$,

$$\Re^\infty_{RM-FF}(\alpha) \le \frac{1}{(c+1)(2^{1/(2+c)}-1)}, \text{for } c = 0,1,2,\ldots$$

When $c \to \infty$, $(c+1)(2^{1/(2+c)}-1) \to \ln 2$. Then $\Re^\infty_{RM-FF} \le 1/\ln 2$. ■

It is clear that if α is small, RM-FF performs well. However, its performance degrades rapidly when $\alpha > 0.4142$. If we can find a better way to schedule the tasks with large utilization, and use the RM-FF to schedule the tasks with small utilization, then the

overall performance of the combined algorithm will be improved. We are thus motivated to develop a new allocation algorithm that is based on the well-known divide-and-conquer strategy. It is called the Refined-Rate-Monotonic-First-Fit (RRM-FF).

RRM-FF first divides the processors into two groups such that within each group there is a semi-infinite number of processors. Then it also divides the task set into two groups according to their utilizations such that tasks within a group are assigned to the same group of processors. Let the processors in the first group (or the P group) be indexed as $P_1, P_2, \ldots$, and processors in the second group (or the Q group) be indexed as $Q_1, Q_2, \ldots$, with each one initially in the idle state. A task τ_i belongs to the first group if its utilization is larger than 1/3, i.e., $u_i > 1/3$, otherwise it belongs to the second group. The tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ will be scheduled in that order. To schedule τ_i, RRM-FF first identifies the task group it belongs to and then find the least j such that task τ_i, together with all the tasks that have been assigned to processor P_j (or Q_j), can be feasibly scheduled, and assign task τ_i to P_j. The First-Fit heuristic is used to assign tasks in both groups.

RRM-FF can be described in a more algorithmic format as follows: i denotes the ith task, m_P and m_Q the numbers of processors allocated in the first and second processor groups, respectively.

Refined-RM-First-Fit (RRM-FF) (Input: task set Σ; Output: m)

(1) $i := 1$; $m_P := 1$; $m_Q := 1$;
(2) **If** $(u_i > 1/3)$ **Then** {
(3) $j := 1$; feasible := **False**; // $u_i > 1/3$. To be assigned to processor group P
(4) **Repeat**
(5) **If** $(k_{P,j} = 0)$ **OR** $(k_{P,j} = 2)$**Then** feasible:=**True**
(6) **Else** $\{min := i$; $max := j$; **If** $(T_j > T_i)$ **Then** $\{min := j$; $max := i$; $\}$;
(7) **If** $(\lfloor T_{\max}/T_{\min} \rfloor C_{\min} + C_{\max} \leq \lfloor T_{\max}/T_{\min} \rfloor T_{\min})$ OR $(\lceil T_{\max}/T_{\min} \rceil$ $C_{\min} + C_{\max} \leq T_{\max})$ **Then** feasible := **True**
(8) **Else** $j := j + 1$;
(9) }
(10) **Until** (feasible);
(11) $k_{P,j} = k_{P,j} + 1$; $U_{P,j} = U_{P,j} + u_i$; **If** $(j > m_P)$ **Then** $m_P := j$;
(12) }
(13) **Else** { // $u_i > 1/3$. To be assigned to processor group Q
(14) $j := 1$; **While** $(u_i > \frac{2}{\prod_{l=1}^{k_{Q,j}} u_{Q,j,l}} - 1)$ **Do** $j := j + 1$;
(15) $k_{Q,j} := k_{Q,j} + 1$; $U_{Q,j} := U_{Q,j} + u_i$; **If** $(j > m_Q)$ **Then** $m_Q := j$;
(16) }
(17) $i := i + 1$; **If** $(i > n)$ **Then** $\{m := m_P + m_Q$; Exit;$\}$ **Else Goto** 2;

When the algorithm returns, m is the total number of processors required to execute a given set of tasks. Note that the grouping of tasks is "imaginary", and clearly the algorithm is on-line. Also note that the schedulability condition used for scheduling tasks in the first group is the IFF condition. Since at most two tasks can be assigned to any processor in the first processor group P, the IFF schedulability test can be reduced to

Table 3. The Worst-Case Performance Bounds of RRM-FF under α

α	> 0.4142	< 0.4142	< 0.2599	< 0.1892	< 0.1487	< 0.02
$\Re_A^\infty$	2.00	1.92	1.76	1.68	1.63	1.47

just two comparison operations. Therefore, the overall time complexity of the RRM-FF algorithm is still $O(n \log n)$. The worst-case performance bound for RRM-FF is given in Theorem 8. The following lemma, recently obtained in (Burchard, 1994), was key to the proof of the upper bound.

LEMMA 7 *If n tasks cannot be feasibly scheduled on processors using the IFF condition, then the total utilization of the n tasks is greater than $\frac{n}{1+2^{1/n}}$.*

THEOREM 8 *Let N and N_0 be the number of processors required by RRM-FF and the minimum number of processors required to feasibly schedule a given set of tasks, respectively. Then $\Re_{RRM-FF}^\infty \leq 2.0$. The upper bounds of RRM-FF with regard to α are given in Table 3 for a few values of α.*

Proof: Let $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n tasks, with their utilizations $\{u_1, u_2, \ldots, u_n\}$,respectively. Then the total utilization of the task set is given by $\sum_{i=1}^{n} u_i$. Suppose that N_P and N_Q processors are used by RRM-FF to schedule the task set Σ, where N_P and N_Q are the numbers of processors allocated in processor group P and processor group Q, respectively. Then $N = N_P + N_Q$. Among the N_P processors, let n_1 be the number of processors assigned one task and n_2 be the number of processors assigned two tasks. Then $N_P = n_1 + n_2$.

In the processor group P, for the n_1 processors to each of which one task is assigned, $\sum_{i=1}^{n_1} u_i > \frac{n_1}{1+2^{1/n_1}} > \frac{n_1}{2} - \frac{\ln 2}{4}$. For the n_2 processors to each of which two tasks are assigned, $\sum_{i=1}^{n_2}(u_{i,1} + u_{i,2}) > \frac{2n_2}{3}$ since $u_{i,1} > 1/3$ and $u_{i,2} > 1/3$.

In the processor group Q, among all N_Q processors, each of them must be assigned at least two tasks except possibly for the last processor, since $u_i \leq 1/3$. According to Lemma 4, among all processors, on each of which at least two tasks are assigned, there are at most one processor whose utilization is less than or equal to $2(2^{1/3} - 1)$, then we have

$$\sum_{i=1}^{N_Q} u_i \geq (N_Q - 1)2(2^{1/3} - 1) > \frac{N_Q - 1}{2}.$$

Since $\sum_{i=1}^{n_1} u_i > \frac{n_1}{2} - \frac{\ln 2}{4}$, $\sum_{i=1}^{n_2}(u_{i,1} + u_{i,2}) > \frac{2n_2}{3}$, $\sum_{i=1}^{N_Q} u_i > \frac{N_Q - 1}{2}$, we have

$$\sum_{i=1}^{n} u_i = \sum_{i=1}^{n_1} u_i + \sum_{i=1}^{n_2}(u_{i,1} + u_{i,2}) + \sum_{i=1}^{N_Q} u_i > \frac{N-1}{2} - \frac{\ln 2}{4},$$

i.e., the total utilization of all processors is equal to the total utilizations of all tasks in a task set.

Since $N_0 \geq \sum_{i=1}^{n} u_i$, it follows from above that $2N_0 + 1 + \ln 2/2 \geq N$. Therefore, $\Re_{RRM-FF}^\infty \leq 2$.

For $\alpha \leq 2^{1/2} - 1$, then each processor in the first processor group P must be assigned two tasks. By arguments similar to the above one and the one in the proof of Theorem , we obtain the results that are listed in Table 3. ■

5. Rate-Monotonic-Best-Fit and Its Refinement

When RM-FF schedules a task, it always assigns it to the lowest indexed processor on which the task can be scheduled. This strategy may not be optimal in some cases. For example, the lowest indexed processor on which a task is scheduled may be the one with the largest available utilization among all those busy (non-idle) processors. This processor could have been used to execute a future task with large enough utilization so that it could not be scheduled on any busy processors, had it not been assigned a task with a small utilization earlier on. In order to overcome these likely disadvantages, a new algorithm is designed, which is based on the Best-Fit bin-packing algorithm.

It is a well-known fact that the Best-Fit heuristic has the same worst-case performance bound as the First-Fit (Coffman et al., 1975). Yet we cannot automatically conclude from the bin-packing results that RM-FF and RM-BF will have the same worst-case performance bound, because of the difference between the bin-packing problem (i.e., the classical one-dimensional bin-packing) and the RMMS problem. The major difference is that the sizes of bins in bin-packing are unitary and the utilizations of processors are values ranging from $\ln 2$ to 1 as given by the schedulability condition.

In bin-packing, when an item is allocated by the Best-Fit, the lowest indexed bin in which the item can be fit and whose content is the largest among all the non-empty bins in which the item can be fit, is chosen to contain the item. Since the "sizes" of the bins are unitary, finding the fitting bin whose content is the largest among all the non-empty fitting bins is equivalent to finding the fitting bin whose available space is the smallest among all the non-empty fitting bins. This "equivalence" property of Best-Fit does not hold when Best-Fit is used to schedule tasks on processors. The "unfilled" utilization of a processor is not only determined by the total utilization of the tasks assigned to it, but also by the number of tasks. Therefore, it is possible that the available utilization of a processor with a currently large utilization is larger than that of a processor with a currently small utilization. For example, processor P_1 is currently assigned two tasks, each with a utilization of $(2^{1/3} - 1) \approx 0.259$. Then the total utilization of processor P_1 is $U_1 = 2(2^{1/3} - 1) < 0.52$, and the available utilization of P_1 is $(2^{1/3} - 1)$. For another processor P_2 to which one task with a utilization of $U_2 = 0.52$ is assigned, its available utilization is given by $(1 - 0.52)/(1 + 0.52) > 0.31$. Therefore, the available utilization of processor P_2 is larger than that of processor P_1 even though $U_1 < U_2$. The schedulability condition used in both the calculations is the one in expression (1).

In other words, there are at least two notably different ways in which the Best-Fit heuristic can be applied to allocating tasks to processors: one is to find the "fitting" processor with the largest utilization, and the other is to find the "fitting" processor with the smallest available utilization. Presumably these two variations might have different worst-case performance bounds. In the following, we only investigate one variation of

the Best-Fit strategy, where the "best fit" is the "fitting" processor with the smallest available utilization.

Algorithm RM-BF: Let the processors be indexed as $P_1, P_2, \ldots$, with each initially in the idle state, i.e., with zero utilization. The tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$, which are ordered according to their non-decreasing periods, will be scheduled in that order. To schedule τ_i, find the least j such that task τ_i, together with all the tasks that have been assigned to processor P_j can be feasibly scheduled according to the condition $2(1+u_j/k_j)^{-k_j}-1$ for a single processor, and $2(1+u_j/k_j)^{-k_j}-1$ be as small as possible, and assign task τ_i to P_j, where k_j and U_j are the number of tasks already assigned to processor P_j and the total utilization of the k_j tasks, respectively.

With its "minimal unfilled utilization" strategy in assigning tasks to processors, the RM-BF algorithm does not outperform RM-FF in the worst cases, as shown by Theorem 9. Before we prove the bounds for RM-BF, the following definition is needed.

Definition. For all the processors required to schedule a given set of tasks by RM-BF, they are divided into two types of processors:

Type (I): for all the tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ with utilizations $\{u_1, u_2, \ldots, u_n\}$ that were assigned to a processor P_x in the completed RM-BF schedule, there exists at least one task τ_i with $i \geq 2$ that was assigned to P_x, not because it could not be assigned on any processor P_y with lower index, i.e., $y < x$, but because $2[1+\sum_{l=1}^{i-1} u_{x,l}/(i-1)]^{-(i-1)} - 1 < 2(1+\sum_{l=1}^{n_y} u_{y,l}/n_y)^{-n_y} - 1$, where n_y is the number of tasks assigned to processor P_y. Processor P_x is called a Type (I) processor. Such a task τ_i is, for convenience, referred to as a task with Type (I) property.

Type (II): They consist of all the processors that do not belong to Type (I).

The following lemma follows immediately from Lemma 3.

LEMMA 8 *If n tasks cannot be feasibly scheduled on $n-1$ processors according to RM-BF, then the total utilization of the set of tasks is greater than $n(2^{1/2}-1)$.*

LEMMA 9 *In the completed RM-BF schedule, if the mth task on any of the Type (I) processors has Type (I) property, where $m \geq 2$, then the total utilization of the first $(m-1)$ tasks on that processor is greater than $(m-1)(2^{1/m}-1)$.*

Proof: Let $\{\tau_{k,1}, \tau_{k,2}, \ldots, \tau_{k,m-1}\}$ be the tasks that were assigned a processor P_k of Type (I), and P_y, with $y < k$, is one of the processors on which τ_m could have been scheduled, but $2[1+\sum_{l=1}^{m-1} u_{k,l}/(m-1)]^{-(m-1)} - 1 < 2(1+\sum_{l=1}^{n_y} u_{y,l}/n_y)^{-n_y} - 1$, where n_y is the number of tasks assigned to processor P_y, and where $u_{x,l}$ is the utilization of task $\tau_{x,l}$ on processor P_x.

Since $u_{k,i} > 2(1+\sum_{l=1}^{n_y} u_{y,l}/n_y)^{-n_y} - 1$ (note that this is true even though $\tau_{k,i}$ is assigned to processor P_k before some of tasks among the n_y tasks are assigned to processor P_y), for $1 \leq i \leq m-1$, it follows that $u_{k,i} > 2(1+\sum_{l=1}^{n_y} u_{y,l}/n_y)^{-n_y} - 1 > 2[1+\sum_{l=1}^{m-1} u_{k,l}/(m-1)]^{-(m-1)} - 1$. Summing up these $(m-1)$ inequalities yields

$$\sum_{j=1}^{m-1} u_{k,j} > 2(m-1)[1+\sum_{l=1}^{m-1} u_{k,l}/(m-1)]^{-(m-1)} - (m-1).$$

Solving the above equation yields

$$\sum_{j=1}^{m-1} u_{k,j} > (m-1)(2^{1/m}-1).$$

The following lemma is key to the proof of Theorem 9.

LEMMA 10 *In the completed RM-BF schedule, among the processors of Type (I) on which the second task has Type (I) property, there are at most three of them, each of which has a total utilization less than* $2(2^{1/3}-1)$.

Proof: This lemma is proven by contradiction. Let P_i, P_j, P_k, and P_l be the four processors, each of which has a total utilization less than $2(2^{1/3}-1)$ with $i<j<k<l$, i.e.,

$$\sum_{x=1}^{n_i} u_{i,x} < 2(2^{1/3}-1)$$

$$\sum_{x=1}^{n_j} u_{j,x} < 2(2^{1/3}-1)$$

$$\sum_{x=1}^{n_k} u_{k,x} < 2(2^{1/3}-1)$$

$$\sum_{x=1}^{n_l} u_{l,x} < 2(2^{1/3}-1)$$

where $n_i \geq 2$, $n_j \geq 2$, $n_k \geq 2$, and $n_l \geq 2$ are the number of tasks assigned to processors P_i, P_j, P_k, and P_l, respectively.

Let's define $u_{i,1}$ and $u_{i,2}$ to be the utilizations of the first task $\tau_{i,1}$ and second task $\tau_{i,2}$ assigned to processor P_i, $u_{j,1}$ and $u_{j,2}$ to be the utilizations of the first task $\tau_{j,1}$ and second task $\tau_{j,2}$ assigned to processor P_j. $u_{k,1}$ and $u_{k,2}$, $u_{l,1}$ and $u_{l,2}$ are similarly defined. We further assume that n_y is the number of tasks which have been assigned to processor P_i, when the second task on processor P_j is assigned. Note that $i<j$ and $1 \leq n_y \leq n_j$.

There are three cases to consider.

Case 1: Tasks $\tau_{j,1}$ and $\tau_{j,2}$ are assigned to processor P_j after task $\tau_{i,2}$ is assigned to processor P_i. Since task $\tau_{j,2}$ is a Type (I) task, the following inequality must hold

$$2(1+u_{j,1})^{-1} - 1 < 2[1+\sum_{x=1}^{n_y} u_{i,x}/n_y]^{-n_y} - 1.$$

Note that $n_y \geq 2$, i.e., other tasks may have been assigned to processor P_i after task $\tau_{i,2}$ but before $\tau_{j,1}$ is assigned to processor P_j.

Since $2[1+\sum_{x=1}^{n_y} u_{i,x}/n_y]^{-n_y} - 1 \leq 2[1+(u_{i,1}+u_{i,2})/2]^{-2} - 1 < 2(1+u_{i,1}/2)^{-2} - 1$, we have

$$2(1+u_{j,1})^{-1} - 1 < 2(1+u_{i,1}/2)^{-2} - 1,$$

i.e., $1+u_{j,1} > (1+u_{i,1}/2)^2$.

Case 2: Tasks $\tau_{j,1}$ and $\tau_{j,2}$ are assigned to processor P_j after task $\tau_{i,1}$ is assigned to processor P_i but before task $\tau_{i,2}$ is assigned to processor P_i.

This case is impossible with RM-BF scheduling. Since $\sum_{x=1}^{n_i} u_{i,x} < 2(2^{1/3}-1)$ and $u_{i,1} > (2^{1/2}-1)$ according to Lemma 9, $u_{i,2} < 2(2^{1/3}-1) - (2^{1/2}-1) \approx 0.1056$. Since task $\tau_{j,2}$ is assigned to processor P_j before task $\tau_{i,2}$ is assigned to processor P_i, and task $\tau_{j,2}$ is a Type (I) task, $2(1+u_{i,1})^{-1} - 1 > 2(1+u_{j,1})^{-1} - 1$, i.e.,

$$u_{i,1} < u_{j,1}. \tag{21}$$

Since task $\tau_{i,2}$ is also a Type (I) task, it must be true according to the definition that

$$2(1+u_{i,1})^{-1} - 1 < 2(1+\sum_{x=1}^{n_z} u_{j,x}/n_z)^{-n_z} - 1,$$

where n_z is the number of tasks that have been assigned to processor P_j after task $\tau_{j,2}$, but before task $\tau_{i,2}$ is assigned to processor P_i. Note that it is conceivable that other tasks may have been assigned to processor P_j after task $\tau_{j,2}$ but before task $\tau_{i,2}$ is assigned to processor P_i.

Since $2(1+u_{i,1})^{-1} - 1 < 2(1+\sum_{x=1}^{n_z} u_{j,x}/n_z)^{-n_z} - 1 < 2(1+u_{j,1})^{-1} - 1$, we have $u_{i,1} > u_{j,1}$. This is a contradiction to equation (21).

Case 3: Task $\tau_{j,1}$ is assigned to processor P_j after task $\tau_{i,1}$ is assigned to processor P_i, and task $\tau_{j,2}$ is assigned to processor P_j after task $\tau_{i,2}$ is assigned to processor P_i. Since task $\tau_{j,2}$ is a Type (I) task, the following inequality must hold

$$2(1+u_{j,1})^{-1} - 1 < 2(1+\sum_{x=1}^{n_y} u_{i,x}/n_y)^{-n_y} - 1.$$

Note that $n_y \geq 2$, i.e., other tasks may have been assigned to processor P_i after task $\tau_{i,2}$ but before $\tau_{j,2}$ is assigned to processor P_j.

Since $2(1+\sum_{x=1}^{n_y} u_{i,x}/n_y)^{-n_y} - 1 \leq 2[1+(u_{i,1}+u_{i,2})/2]^{-2} - 1 < 2(1+u_{i,1}/2)^{-2} - 1$, we have $2(1+u_{j,1})^{-1} - 1 < 2(1+u_{i,1}/2)^{-2} - 1$, i.e., $1+u_{j,1} > (1+u_{i,1}/2)^2$.

Therefore for processors P_i and P_j, we have

$$1+u_{j,1} > (1+u_{i,1}/2)^2 \tag{22}$$

For the tasks assigned on processors P_j and P_k, and P_k and P_l, it can be similarly proven that

$$1 + u_{k,1} > (1 + u_{j,1}/2)^2 \tag{23}$$

$$1 + u_{l,1} > (1 + u_{l,1}/2)^2 \tag{24}$$

Summing up equations (22), (23), and (24) yields $u_{l,1} > (u_{i,1}^2 + u_{j,1}^2 + u_{k,1}^2)/4 + u_{i,1}$. Since $u_{i,1} > (2^{1/2} - 1)$, $u_{j,1} > (2^{1/2} - 1)$, and $u_{k,1} > (2^{1/2} - 1)$ according to Lemma 9, $u_{l,1} > 3(2^{1/2} - 1)^2/4 + (2^{1/2} - 1) = 0.5429 > 2(2^{1/3} - 1)$. This results in a contradiction to the assumption that $\sum_{x=1}^{n_l} u_{l,x} < 2(2^{1/3} - 1)$. ■

THEOREM 9 *Let N and N_0 be the number of processors required by RM-BF and the minimum number of processors required to feasibly schedule a given set of tasks, respectively. Then* $2.2833 \leq \lim_{N_0 \to \infty} N/N_0 \leq 2 + (3 - 2^{3/2})/a \approx 2.33$, *where* $a = 2(2^{1/3} - 1)$.

Proof: Similar to what we have done in section 3 for the RM-FF algorithm, we use the same weighting function to map the utilization of a task into the real interval (0, 1]. Note that all the relevant lemmas in section 3 hold for those processors of Type (II) in the RM-BF schedule.

Let $\Sigma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n tasks, with their utilizations $u_1, u_2, \ldots, u_n$, respectively, and $\omega = \sum_{i=1}^{n} f(u_i)$. By Lemma 5, $\omega \leq N_0/a$, where $a = 2(2^{1/3} - 1)$.

Suppose that among the N number of processors used by RM-BF to schedule a given set Σ of tasks, M of them belongs to processors of Type (I). Since all processors of Type (I) must be assigned at least two tasks, there exists for each processor at least an number m with $m \geq 2$ such that the mth task is a Type (I) task. For all the processors of Type (I) on each of which the mth task is a Type (I) task with $m \geq 3$, $\sum_j f(u_j) > 1$ since $\sum_j u_j > 2(2^{1/3} - 1)$ according to Lemma 9.

When $m = 2$, there are at most three of them, each of which has a total utilization less than $2(2^{1/3} - 1)$. Therefore, for all the processors of Type (I), there are at most three processors whose $\sum_j f(u_j)$ is less than 1 in the RM-BF schedule.

Now let $L = n_1 + n_2$ be defined similarly as in section 3, except that they are for processors of Type (II). All the results derived in section 3 are applicable to the set of Type (II) processors in the RM-BF schedule. Since $\sum_{i=1}^{n} f(u_i) = \omega \leq N_0/a$ and

$$\sum_{i=1}^{n} f(u_i) \geq (N - L - M) + n_1(2^{1/2} - 1)/a = N - n_1 - n_2 + n_1(2^{1/2} - 1)/a -$$

$$\geq N - 2N_0[1 - (2^{1/2} - 1)/a] - n_2 - 3,$$

we have

$$\lim_{N_0 \to \infty} N/N_0 \leq [2a + 1 - 2(2^{1/2} - 1)]/a \approx 2.33,$$

where $a = 2(2^{1/2} - 1)$.

The lower bound is proven by repeating the same argument in Theorem 6 for RM-BF. Therefore, $2.2833 \leq \lim_{N_0 \to \infty} N/N_0$. ■

Note that even though RM-BF has the same worst-case performance bound as RM-FF, special cases exist where RM-BF performs better than RM-FF, and vice versa. For example, for a set of four tasks with their utilizations given as follows, two processors are needed by RM-BF while three processors are required by RM-FF to schedule it.

$u_1 = 2/5$, $u_2 = 3/7 + \epsilon$, $u_3 = (4 - 7\epsilon)/(10 + 7\epsilon)$, and $u_4 = 3/7$ for arbitrarily small $\epsilon > 0$.

Let α be the maximum allowable utilization of a task, i.e., $\alpha = \max_i\{C_i/T_i\}$. Then we can prove, similar to what we have done in section 4 that when α is small, the worst-case performance of RM-BF can be significantly improved, as stated in Theorem 10. Based on similar observation, we can modify RM-BF to develop a new algorithm called Refined-Rate-Monotonic-Best-Fit (RRM-BF) to cope with situations where α is large.

RRM-BF works as follows: it divides the processors into two groups such that within each group there is a semi-infinite number of processors. It also divides the task set into two groups according to their utilizations such that tasks within a group are assigned to the same group of processors. A task τ_i belongs to the first group if its utilization is larger than 1/3, i.e., $u_i > 1/3$, otherwise it belongs to the second group. The tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ will be scheduled in that order. To schedule τ_i, RRM-BF first identifies the task group it belongs to, and then find the least j such that task τ_i, together with all the tasks that have been assigned to processor P_j (or Q_j), can be feasibly scheduled, and assign task τ_i to P_j. The Best-Fit heuristic is used to assign tasks in both groups. The schedulability condition used to schedule tasks with utilizations less than or equal to 1/3 is $2(1 + U_j/k_j)^{-k_j} - 1$, where k_j and U_j are as defined above. The schedulability condition used to schedule tasks with utilizations larger than 1/3 is the necessary and sufficient condition.

The major result concerning RRM-BF is stated in Theorem 11. The proof of both Theorem 10 and Theorem 11 is left as an exercise for the reader.

THEOREM 10 *Let N and N_0 be the number of processors required by RM-BF and the minimum number of processors required to feasibly schedule a given set of tasks, respectively. Further let $\alpha = \max_{1 \leq i \leq n}\{u_i\}$. If $\alpha \leq 2^{1/(1+c)} - 1$, then $\Re^{\infty}_{RM-BF}(\alpha) \leq \frac{1}{(c+1)(2^{1/(2+c)}-1)}$, for $c = 0, 1, 2, \ldots$ When $c \longrightarrow \infty$, $(c+1)(2^{1/(2+c)}-1) \longrightarrow \ln 2$. Then $\Re^{\infty}_{RM-BF} \leq \frac{1}{\ln 2}$. The upper bounds of RM-BF with regard to α are given in Table 2 for a few values of α.*

THEOREM 11 *Let N and N_0 be the number of processors required by RRM-BF and the minimum number of processors required to feasibly schedule a given set of tasks, respectively. Then $\Re^{\infty}_{RRM-BF} \leq 2$. The upper bounds of RRM-BF with regard to α are given in Table 3 for a few values of α.*

6. Empirical Studies of the New Algorithms

In this study, the performance bounds of the new algorithms were derived under worst-case assumptions. While a worst-case analysis assures that the performance bounds are satisfied for some large task set, it does not provide insight into the average-case behavior of the algorithms. To obtain the average-case performance of the new algorithms, one can analyze the schemes with probabilistic assumptions, or conduct simulation experiments to empirically study the average-case performance. Since a probabilistic analysis of our algorithms is beyond the scope of this study, we resort to simulation to gain insight into the average-case behavior of the new algorithms.

We present simulation experiments for large task sets with $100 \leq n \leq 1000$ tasks. In each experiment, we vary the value of parameter α—the maximal load factor of any task in the set, i.e., $\alpha = \max_i\{C_i/T_i\}$. The task periods are assumed to be uniformly distributed with values $1 \leq T_i \leq 500$. The run-times of the tasks are also taken from a uniform distribution with range $1 \leq C_i \leq \alpha T_i$. The performance metric in all experiments is the number of processors required to execute a given task set. We compare the new schemes with the existing scheme NextFit-M (NF-M) in the literature. All assignment schemes are executed on identical task sets.

Since an optimal schedule cannot be calculated for large task sets, we use the total utilization or load ($U = \sum_{i=1}^{n} C_i/T_i$) of the task set as the lower bound for the number of processors required.

The outcome of the simulation experiments is shown in Figure 2 and Figure 3. The maximum utilization of a task is set to α = 0.2, 0.4, 0.6, 0.8 in both set of experiments. Each data point depicts the average value of *20* independently generated task sets with identical parameters. Note that our four algorithms consistently outperforms NF-M. In Figure 2, the performance of RM-FF and RM-BF is almost the same. We therefore only plot the performance of RM-FF in Figure 3 for comparison. Also the performance of RM-FF and RRM-FF (or RM-BF and RRM-BF) cannot be distinguished since they are identical when $\alpha < 1/3$. As we increase the value of α from 1/3 to 1, we observe that the performance of RRM-FF and RRM-BF improves and then degrade a little bit, compared to that of RM-FF. All results show that the number of processors required for each algorithm increases proportionally to the total utilization of the task set.

7. Conclusions

In this paper, we investigated the problem of scheduling a set of periodic tasks on a multiprocessor system so as to minimize the number of processors used. Four on-line algorithms were developed using a newly derived schedulability condition. The worst-case analysis shows that both algorithms RRM-FF and RRM-BF have a worst-case performance bound of 2. The performance of the algorithms can be significantly improved when the maximum allowable utilization of a task is small. This worst-case bound is the currently the best for on-line allocation algorithms for solving the same problem. It is also equal to the currently known best bound for off-line algorithms. Simulation is conducted to assess the average-case behavior of the algorithms. Experimental results

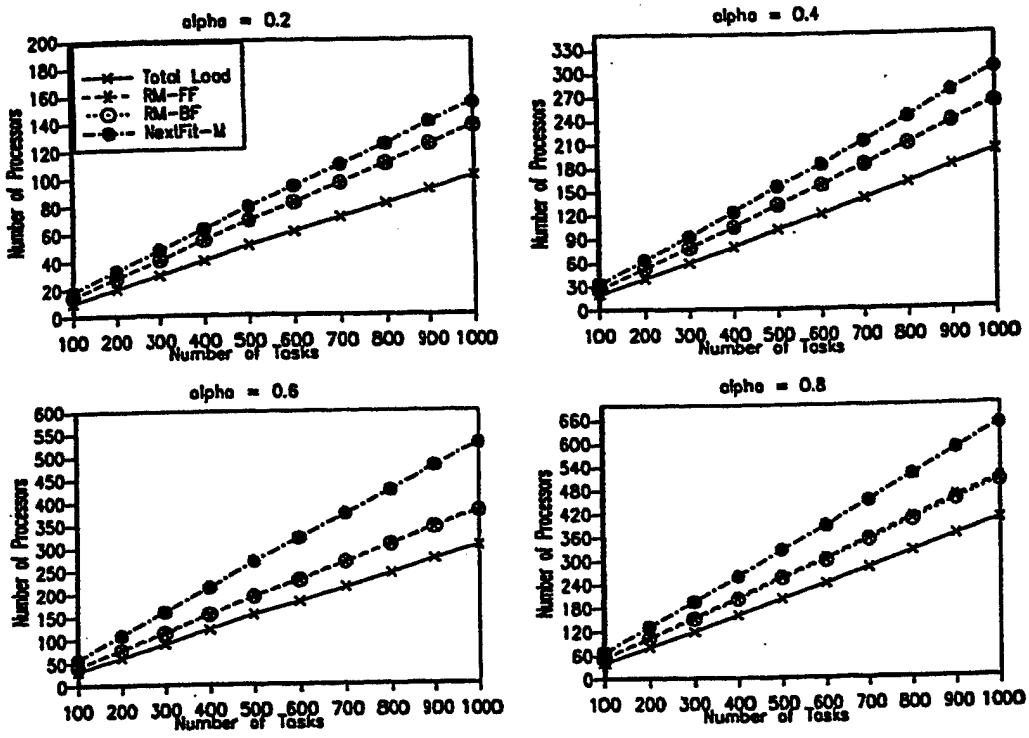


Figure 2. Average-case Performance of RM-FF(BF) and NF-M.

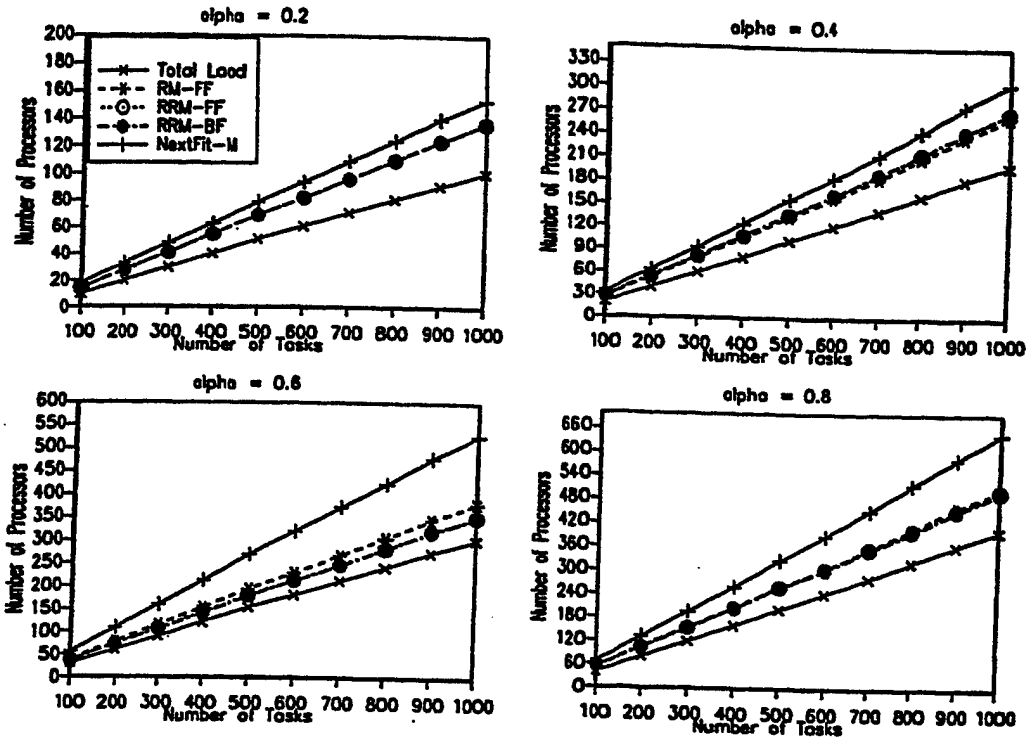


Figure 3. Average-case Performance of RRM-FF(BF), RM-FF, and NF-M.

indicate that the four algorithms consistently outperform the existing on-line algorithm NF-M.

There are a number of places where further improvement is possible. The first interesting question is what the exact bounds of the heuristics are if the necessary and sufficient condition—the IFF condition is used instead in scheduling tasks. The second question is whether there exist some off-line algorithms whose worst-case performance is better than the currently known bound of 2. Finally, it is interesting to find out whether a variation of the Best-Fit heuristic other than the one we investigated in this paper has a better worst-case performance.

Finally we want to remark that there is usually a time lapse between the discovery of a new theory and its practical application. The Rate-Monotonic Scheduling was first discovered in 1972-1973 by Liu and Layland, and Serlin. It took about 15 years when the Rate-monotonic algorithm was actually supported by a real-time operating system. Now the Rate-monotonic algorithm has been used widely in a number of applications. The first result on Rate-Monotonic Multiprocessor Scheduling (RMMS) was derived around 1977-1978 by Dhall and Liu. Interests in real-time task scheduling on multiprocessor systems have been increasing rapidly, due to the inevitable employment of multiprocessors in many real-time systems. The results obtained in this paper are timely results for the research community and for practitioners at large.

Acknowledgments

We would like to thank the referees for their many suggestions, which improve the presentation of this paper. This work was supported in part by ONR, by CIT, and by Loral Federal Systems.

Appendix

In this appendix, we show that some errors exist in the proof for the upper bound of RMFF by Dhall and Liu in (Dhall and Liu, 1978). Their RMFF is almost the same as our RM-FF except that in their RMFF, the IP condition is used with the tasks being sorted in the order of increasing period. They obtained the following results:

Lemma I: If n tasks cannot be feasibly scheduled on $n-1$ processors according to RMFF, then the utilization factor of the set of tasks is greater than $n/(1+2^{1/3})$.

Lemma II: If tasks are assigned to the processors according to RMFF, among all processors to each of which two tasks are assigned, there is at most one processor for which the utilization factor of the set of the two tasks is less than 1/2.

Theorem I: Let N be the number of processors required to feasibly schedule a set of tasks by RMFF, and N_0 the minimum number of processors required to feasibly schedule the same set of tasks. Then as N_0 approaches infinity, $2 \leq N/N_0 \leq 4\times 2^{1/3}/(1+2^{1/3})(\cong 2.23)$.

Unfortunately, Lemma I is incorrect, as shown by the following counter example. Lemma II gives a weak result for RMFF. These two errors led the authors to arrive at

the wrong upper bound. In the following, we first show the incorrectness of Lemma I, and then give a strong version of Lemma II.

Example: consider the case where $m = 2$ and the two tasks are given as follows:
$\tau_1 = (2^{1/2} - 1, 1)$,
$\tau_2 = (2 - 2^{1/2} + \epsilon, 2^{1/2})$, where ϵ is a small number and $\epsilon > 0$.

According to RMFF, τ_1 is first assigned to a processor. Since $u_1 = 2^{1/2} - 1$ and $2(1+u_1)^{-1} - 1 = 2^{1/2} - 1 < 2^{1/2} - 1 + \epsilon/2^{1/2} = u_2$, τ_2 cannot be scheduled together with task τ_1 on one processor, according to the IP condition. Since τ_1 and τ_2 cannot be scheduled on one processor, $u_1 + u_2$ must be greater than $2/(1+2^{1/3}) \cong 0.88$, according to Lemma I. But $u_1 + u_2 = 2(2^{1/2} - 1) + \epsilon/2^{1/2} = 0.8284 + \epsilon/2^{1/2}$, which is less than 0.88 for small ϵ.

A stronger (tight) version of their Lemma II can be given in the following lemma. The proof is similar to that of Lemma 4.

Lemma II (Revised): If tasks are assigned to the processors according to RMFF, among all processors to each of which two tasks are assigned, there is at most one processor for which the utilization factor of the set of the two tasks is less than $2(2^{1/3}-1)$.

Note that Lemma I is true if their RMFF used instead the necessary and sufficient condition given by Joseph and Pandya in their 1986 paper (Joseph and Pandya, 1986) or by Lehoczky et al in their 1989 paper (Lehoczky et al., 1989) for $m > 2$, and our new result as stated in Lemma 7. It may be the case that Dhall and Liu indeed considered the problem of scheduling a set of n tasks on n processors (with one task on each processor), and obtained the result in Lemma I. However, for their upper bound to hold, the necessary and sufficient condition must be used in their RMFF scheme instead. In either case, the upper bound of 2.23 does not fit.

References

Burchard, A. A., J. Liebeherr, Y. Oh, and S.H. Son (1994). Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems, submitted for publication, January 1994.

Coffman, E.G.JR. (ED.) (1975). Computer and Job Shop Scheduling Theory, New York: Wiley, 1975.

Coffman,E.G.JR., M.R. Garey, and D.S. Johnson (1985) Approximate Algorithms for Bin Packing - An Updated Survey, In Algorithm Design for Computer System Design, (49-106) G. AUSIELLO, M. LUCERTINIT, and P. SERAFINI (Eds), Springer-Verlag, New York, 1985.

Davari, S. and S.K. Dhall (1986). An On Line Algorithm for Real-Time Tasks Allocation, IEEE Real-Time Systems Symposium, 194-200 (1986a).

Davari, S. and S.K. Dhall (1986). On a Periodic Real-Time Task Allocation Problem, Proc. of 19th Annual International Conference on System Sciences, 133-141 (1986b).

Dhall, S..K. and C..L. Liu (1978). On a Real-Time Scheduling Problem, Operations Research, 26:127-140 (1978).

Gafford, J.D. (1991). Rate-Monotonic Scheduling, IEEE Micro, 34-39 (June 1991).

Garey, M.R. and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP- completeness, W.H. Freeman and Company, NY, 1978.

Johnson, D.S. (1993). Near-Optimal Bin Packing Algorithms, Doctoral Thesis, MIT, 1973.

Joseph, M. and P. Pandya (1986). Finding Response Times in a Real-Time System, The Computer Journal, 29(5):390-395, 1986.

Lehoczky, J.P., L. Sha, and Y. Ding (1989). The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior, IEEE Real-Time Symposium, 166-171 (1989).

.ehoczky, J.P., L. Sha, and J.K. Strosnider (1987). Enhanced Aperiodic Responsiveness in Hard Real-time Environments, IEEE Real-Time Systems Symposium, 261-270 (1987).
.ehoczky, J.P. and S. Ramos-Thuel (1992). An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems, IEEE Real-Time Systems Symposium, 110-123 (1992).
.eung, J.Y.T. and J. Whitehead (1982). On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks, Performance Evaluation, 2:237-250 (1982).
.iu, C.L. and J. Layland (1973). Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, JACM, 10(1):174-189 (1973).
)h, Y. and S.H. Son (1993). On-line Task Allocation Algorithms for Hard Real-Time Multiprocessor Systems, Submitted for Publication, October 1993.
Ramamritham, K. (1990). Allocation and Scheduling of Complex Periodic Tasks, International Conference on Distributed Computing Systems, May 1990.
Ramos-Thuel, S. and J.K. Stronider (1991). The Transient Server Approach to Scheduling Time-Critical Recovery Operations, IEEE Real-Time Systems Symposium, 286-295 (1991).
Serlin, P. (1972). Scheduling of Time Critical Processes, Proceedings of the Spring Joint Computers Conference, 40:925-932 (1972).
Sha, L., J.P. Lehoczky, and R. Rajkumar (1986). Solutions for Some Practical Problems in Prioritized Preemptive Scheduling, IEEE Real-Time Systems Symposium, 181-191 (1986).
Sha, L., R. Rajkumar, J.P. Lehoczky, and K. Ramamritham (1989). Mode Change Protocols for Priority-Driven Preemptive Scheduling, Journal of Real-Time Systems, 1(3):244-264 (1989).
Sha, L., R. Rajkumar, and J.P. Lehoczky (1990). Priority Inheritance Protocols: An Approach to Real-Time Synchronization, IEEE Transactions on Computers, 39(9):1175-1185 (1990).
Sha, L., and J.B., Goodenough (1990). Real-time Scheduling Theory and Ada," Computer, 53-66 (April 1990).
Sprunt, B., L. Sha, and J.P. Lehoczky (1989). Aperiodic Task Scheduling for Hard Real-time Systems, Journal of Real-Time Systems, 1:27-60 (1989).
Tindell, K.W., A. Burns, and A.J. Wellings (1992). Mode Change in Priority Pre-emptively Scheduled Systems, IEEE Real-Time Systems Symposium, 100-109 (1992).

Efficiently Supporting Hard/Soft Deadline Transactions in Real-Time Database Systems*

Chang-Gun Lee† Young-Kuk Kim‡ Sang H. Son§
Sang Lyul Min† Chong Sang Kim†

† Dept. of Computer Engineering, Seoul National University, Seoul 151-742, Korea
‡ Dept. of Computer Science, Chungnam National University, Taejon 305-764, Korea
§ Dept. of Computer Science, University of Virginia, Charlottesville, VA 22903-2242, USA

Abstract

As in other application areas, there is an increasing need for managing a large amount of data in the real-time area. This need gave birth to a system called a real-time database system (RTDBS) that provides database operations with timing constraints. One typical timing constraint in an RTDBS is temporal consistency that states that a transaction must read temporally valid data objects. This requires the data objects to be updated repeatedly. This paper proposes three novel schemes that aim to minimize the number of updates of data objects needed for guaranteeing the temporal consistency requirements of transactions with hard-deadlines. The three guarantee schemes differ from each other in the amount of updates and the implementation complexity. This paper also gives a framework for integrating transactions with soft-deadlines into the three guarantee schemes.

1 Introduction

Recently, computer systems are increasingly used for monitoring and controlling time critical systems such as industrial machine control systems and flight control systems [1, 2, 3]. Such computer systems are called real-time systems and they generally have two types of task: hard-deadline and soft-deadline tasks. For hard-deadline tasks, it is required that they be completed by their deadlines. On the other hand, for soft-deadline tasks, the goal is to meet the deadlines of as many of them as possible. Problems related to both types of task have been studied intensively in the real-time area [4, 5, 6, 7].

As in other application areas, there is an increasing need for managing a large amount of data in the real-time area. This need gave birth to a system called a real-time database system (RTDBS) that provides database operations with timing constraints. An RTDBS has features that distinguish it from a traditional database system. First, it manages *real-time data objects* which reflect the rapidly changing real-world (e.g., chamber temperature, aircraft location and speed) in addition to traditional, relatively static data objects. Such real-time data objects become invalid/obsolete as the external real-world changes and thus they should be updated repeatedly to remain valid. Second, transactions in an RTDBS have timing constraints. One type of such timing constraint is a deadline. The deadline of a transaction may be hard-deadline or soft-deadline depending on its functional requirement. Another timing constraint is *temporal consistency*. There are two types of temporal consistency: absolute and relative temporal consistency. The absolute temporal consistency requires that a transaction read temporally valid (i.e., recent enough) data objects and be committed before any of them becomes invalid. On the other hand, the relative temporal consistency requires that update times of data objects read by a transaction be close enough (a more formal definition is given in Section 2).

Adelberg et al. address the problems arising from updates of real-time data objects that are needed so that the data objects are temporally valid when they are read by transactions [8, 9]. They present a number of schemes that aim to efficiently balance update processing and soft-deadline transaction processing. The schemes make best efforts to maximize the number of transactions that are completed by their deadlines and read only temporally valid data objects. However, the problem of ensuring that transactions are completed before any of data read by them becomes invalid is not addressed. To attack the latter problem, Xiong et al. introduce the concept of a *data-deadline* after which some real-time data objects read by a transaction become invalid [10, 11]. This data-deadline concept places a more stringent deadline onto the transaction.

Both of the above two studies handle only soft-deadline transactions. Kim and Son propose a scheme that handles

*This work was supported in part by the Office of Naval Research and National Security Agency.

both hard-deadline and soft-deadline transactions [12, 13]. The scheme guarantees that all the hard-deadline transactions meet their deadlines without violating their temporal consistency requirements. In the scheme, the update period of a real-time data object is set to be less than a half of the valid interval of the object to keep it *always-fresh*. This approach results in a large number of updates whose updated values are not actually used by any transaction. Such *needless updates* consume a large amount of processing time and thus reduce normal processing time for hard-deadline and soft-deadline transactions.

In this paper, we propose three schemes with varying complexities that guarantee the deadlines and temporal consistency requirements of hard-deadline transactions while minimizing update loads. In addition, we give for each proposed scheme a schedulability test and a method for computing spare capacity that can be used to process soft-deadline transactions.

This paper is organized as follows: The next section details the three guarantee schemes. Section 3 explains how to integrate the processing of soft-deadline transactions into the three guarantee schemes. In Section 4, we give the results of our experiments to evaluate the performance of the guarantee schemes. Finally, we conclude this paper in Section 5.

2 Guaranteeing Hard-deadline Transactions

In this section, we describe three new schemes that aim to reduce update loads while guaranteeing both the deadlines and the temporal consistency requirements of hard-deadline transactions. In describing the schemes, we assume the following.

- Each real-time data object x has an *absolute validity interval*, denoted by $avi(x)$, after which its value becomes invalid. Thus, a real-time data object must be updated before its absolute validity interval expires to remain always valid. The time needed to update x is assumed to be $c(x)$.
- There are n periodic hard-deadline transactions denoted by $\tau_1, \cdots, \tau_n$ where τ_1 has the highest priority and τ_n the lowest priority. Each hard-deadline transaction τ_i has the following attributes that are given *a priori*: (1) period p_i, (2) data read set RS_{τ_i} : set of data objects read by τ_i, (3) execution time c_i : a worst case execution time estimate obtained ignoring the time for updating RS_{τ_i}, and (4) deadline d_i.
- There are soft-deadline transactions that arrive aperiodically. Their arrival times cannot be determined *a priori*.
- Every transaction should satisfy both of the following two types of temporal consistency for timing correctness.

1. **Absolute temporal consistency:** *A transaction should read only valid real-time data objects and complete its execution before the absolute validity interval of any of them expires.*
2. **Relative temporal consistency:** *The difference between the commit time of transaction τ and the timestamp (i.e. the latest update time) of any data object in RS_τ should be less than $rvi(\tau)$, which is called the relative validity interval* [13] *of τ. More formally, the condition can be stated as follows.*

$$\forall x \in RS_\tau, t_{commit}(\tau) - t_{stamp}(x) \leq rvi(\tau)$$

where $t_{commit}(\tau)$ and $t_{stamp}(x)$ denote the commit time of τ and the timestamp of x, respectively.

2.1 Guarantee based on On-Demand Update (OD Guarantee)

One possible approach to minimizing the amount of updates is to update real-time data objects only when they are actually accessed by hard-deadline transactions. This scheme, which we call the on-demand (OD) guarantee scheme, guarantees that hard-deadline transactions always read temporally valid data. However, there still remains a need for a transaction to be committed before the absolute validity interval of any of its data objects expires. For this purpose, we adjust the deadline of a hard-deadline transaction τ_i as follows.

$$\dot{d}_i = \min_{j \leq \frac{HP}{p_i}, x \in RS_{\tau_i}} (er_{ij}(x) + avi(x))$$

In the equation, HP is the hyperperiod of hard-deadline transactions and $er_{ij}(x)$ is the earliest possible time when x can be accessed by τ_{ij}[1]. Note that if τ_i is committed before $\dot{d}_i$, its absolute temporal consistency is guaranteed for all the instances of τ_i in the hyperperiod.

Similarly, the relative temporal consistency of τ_i can be enforced by the following new deadline $\bar{d}_i$.

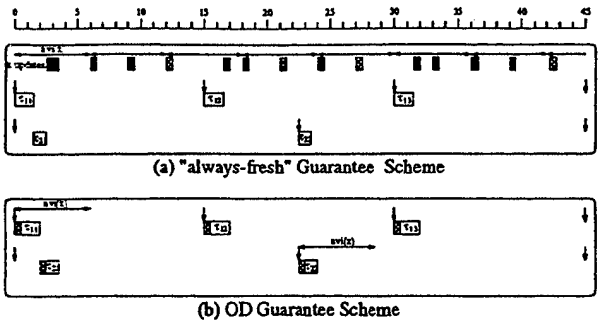
$$\bar{d}_i = \min_{j \leq \frac{HP}{p_i}, x \in RS_{\tau_i}} (er_{ij}(x) + rvi(\tau_i))$$

The above two requirements suggest that the deadline of transaction τ_i be given as follows to satisfy both the absolute and relative temporal consistency in addition to the original deadline d_i, which is given as part of application requirement.

$$d'_i = \min\{\dot{d}_i, \bar{d}_i, d_i\}$$

With this setting, we perform a test to determine whether a given set of hard-deadline transactions is schedulable based on the response time approach by Joseph

[1] $er_{ij}(x)$ for the j-th instance of τ_i can be statically calculated by scheduling the hard-deadline transactions in the hyperperiod assuming their best case execution times.



(a) "always-fresh" Guarantee Scheme

(b) OD Guarantee Scheme

Figure 1: Advantage of the OD guarantee scheme over the always-fresh guarantee scheme

and Pandya [14]. In the OD guarantee scheme, since every instance of τ_i includes the update times of data objects in RS_{τ_i}, the task's execution time is given by $c_i + \sum_{x \in RS_{\tau_i}} c(x)$. This gives the following recursive equation for the worst case response time of τ_i.

$$R_i = c_i + \sum_{x \in RS_{\tau_i}} c(x) + \sum_{j<i} \lceil \frac{R_i}{p_j} \rceil (c_j + \sum_{x \in RS_{\tau_j}} c(x))$$

This worst case response time is compared against d_i' to determine the schedulability of τ_i.

Figure 1 shows the benefits of using the proposed OD guarantee scheme over the guarantee schemes based on *always-fresh* [10, 12, 13, 15]. In the example, we assume that there is a real-time data object x whose absolute validity interval is 6 and it is read by two hard-deadline transactions τ_1 and τ_2 whose periods are 15 and 22.5, respectively. In the *always-fresh* guarantee scheme, every real-time data object is updated periodically with a period smaller than a half of the object's data validity interval to maintain the data object always-fresh. Thus, in our example, the updates of x are made every 3 time units (cf. Figure 1(a)). On the other hand, in the OD guarantee scheme, the updates are made only when x is accessed by some hard-deadline transactions (cf. Figure 1(b)).

In our example, the absolute validity intervals of real-time data objects are small compared to the periods of hard-deadline transactions. This leads to a large number of needless updates in the always-fresh scheme. In the figure, these needless updates are denoted by black rectangles. In the other case where the absolute validity intervals of real-time data objects are large compared to the periods of hard-deadline transactions, the performance of the OD guarantee scheme may be worse than that of the always-fresh scheme. However, since the OD guarantee scheme can be made to update a real-time data object only when it is invalid, the unused update times that result when the data object is up-to-date (i.e., gain time [6]) can be used to process soft-deadline transactions.

2.2 Guarantee based on Periodic Update (PU Guarantee)

In this subsection, we describe another guarantee scheme, called the periodic update (PU) guarantee scheme, which is based on periodic update of real-time data objects similar to the always-fresh scheme. It differs from the always-fresh scheme in that the update periods are set to the maximum values that do not violate the temporal consistency requirements of hard-deadline transactions.

Let $p(x)$ be the update period of x. Then, the intervals where x is valid are given as follows.

$$((t-1)p(x), (t-1)p(x) + avi(x)) \qquad t = 1, 2, 3, \cdots$$

To guarantee the absolute temporal consistency requirements of hard-deadline transactions that read x, each transaction should start and end within one of the intervals. More formally, this condition can be stated as follows.

$$\forall i, j, \forall x \in RS_{\tau_i}, \exists t, ((j-1)p_i, (j-1)p_i + d_i) \subseteq ((t-1)p(x), (t-1)p(x) + avi(x))$$

In addition, to guarantee the relative temporal consistency requirements, the difference between the timestamp of x and the latest possible commit time of a transaction τ_i that reads x should be less than the relative validity interval of τ_i. More formally,

$$\forall i, j, \forall x \in RS_{\tau_i}, \exists t, (j-1)p_i + d_i - (t-1)p(x) \leq rvi(\tau_i)$$

By setting the update period of each real-time data object to the maximum value that satisfies the above two conditions, the total update load can be reduced compared to the always-fresh scheme.

Figure 2 compares the proposed PU guarantee scheme with the always-fresh guarantee scheme assuming the same settings as in Figure 1. In the example, the update period of x in the PU guarantee scheme is given as 7.5 that satisfies both of the above two conditions. From our discussion, it is clear that the PU guarantee scheme performs at least as good as the always-fresh guarantee scheme. In the worst case, the performance of the PU guarantee scheme converges to that of the always-fresh scheme.

2.3 Guarantee based on Aperiodic Update (AU Guarantee)

Even in the PU guarantee scheme, some needless updates are inevitable due to its periodic nature. To rectify this problem, this subsection proposes a guarantee scheme called the aperiodic update (AU) guarantee scheme where updates of real-time data objects are made aperiodically. The key problem in the AU guarantee scheme is how to determine the exact positions in the hyperperiod where updates are necessary to guarantee the temporal consistency requirements of hard-deadline transactions.

The update points of real-time data object x that are needed to guarantee the absolute temporal consistency can

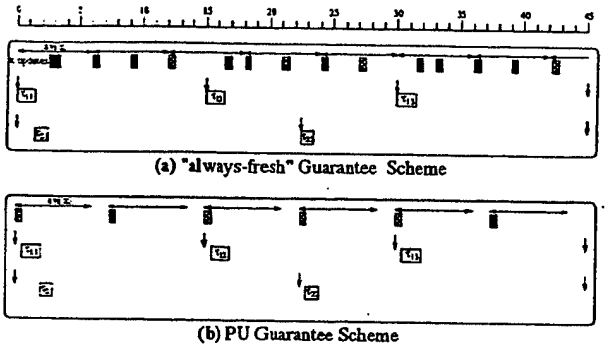


Figure 2: Example of the PU guarantee scheme

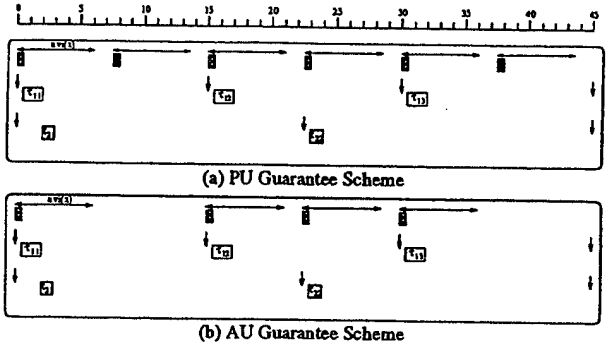


Figure 3: Example of the AU guarantee scheme

be located iteratively as follows: The first update is made at time 0 and the next update is made just before the arrival of a transaction whose arrival time is earliest among the hard-deadline transactions with commit times that are beyond the validity interval of the first update. Other update points are located similarly. In addition, updates are needed to guarantee the relative temporal consistency. The points of these updates can be determined similarly to the case of absolute temporal consistency. In the actual implementation, update points for both absolute and relative temporal consistency are determined in a single pass to eliminate redundant updates that might occur if the two temporal consistency requirements are addressed separately.

In the AU guarantee scheme, the priority of an aperiodic request for updating x is initially set to the lowest possible priority. However, when a hard-deadline transaction that reads x arrives, the update request inherits the priority of that transaction. With this priority inheritance, the worst case response time of τ_i can be given as follows.

$$R_i = c_i + \sum_{j<i} \lceil \frac{R_i}{p_j} \rceil c_j + \sum_{x \in RS_{\tau_j}, j \leq i} k(x, R_i) c(x)$$

where $k(x, R_i)$ is the number of update points of x in interval $[0, R_i]$.

Figure 3 compares the AU guarantee scheme with the PU guarantee scheme assuming the same settings as previously. From the example, we can note that the AU guarantee scheme eliminates all the needless updates that previously existed in the PU guarantee scheme. This is made possible by allowing aperiodic update of real-time data objects.

3 Scheduling of Soft-deadline Transactions

Soft-deadline transactions are processed using spare processing capacity left after guaranteeing hard-deadline transactions. Many schemes that have been proposed to make use of this spare processing capacity [5, 6, 16, 17] can be applied to our three guarantee schemes with minor changes. In this paper, we concentrate on the dynamic slack stealing approach by Davis et al. [6]. This approach gives a method for computing the slack time $S_{\tau_i}(t)$, which is the spare time that is available at time t for processing aperiodic tasks without violating the deadline of a hard-deadline task τ_i. This method can be used to compute the maximum time that can be provided for soft-deadline task processing at time t without violating the deadline of any hard-deadline task (in particular this maximum time is given as $\min_i S_{\tau_i}(t)$ and for more details of the dynamic slack stealing algorithm, refer to [6]).

This dynamic slack stealing can be directly applied to the OD and PU guarantee schemes since all the tasks in these schemes including those for updating real-time data objects are periodic. For the AU guarantee scheme, the slack value $S_{\tau_i}(t)$ of τ_i should be reduced by the amount of processing time for updates that have been pre-scheduled to occur before the next deadline of τ_i.

When soft-deadline transactions are scheduled using slacks, their processing and additional updates for their temporal consistency should be balanced so that as many of them as possible can be serviced before their deadlines. For this purpose, the guarantee schemes proposed in this paper can make use of the balancing schemes explained in [8, 9].

4 Experimental Results

In this section, we present the results of our simulation study to compare the performance of the guarantee schemes proposed in this paper. Only the always-fresh, OD, and AU guarantee schemes are evaluated, since we are still working on the PU guarantee scheme. In addition, we explore only the effects of absolute temporal consistency requirements since we expect that the relative temporal consistency requirements have similar effects on

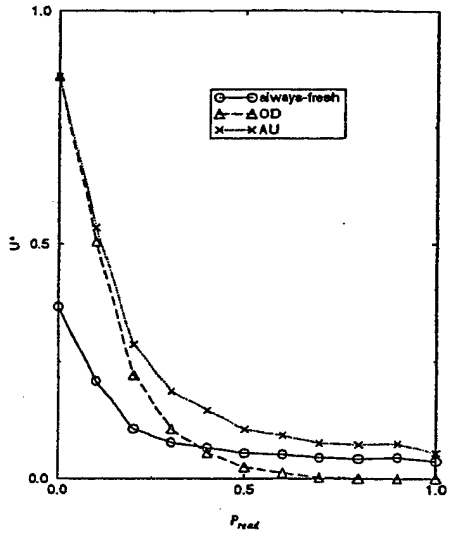


Figure 4: Breakdown utilization according to P_{read} ($L_{avi} = 0.5$)

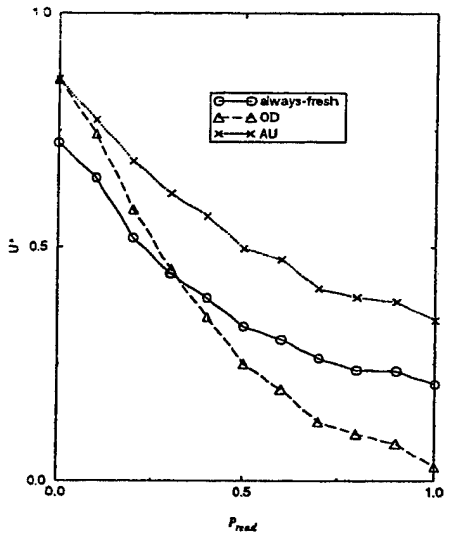


Figure 5: Breakdown utilization according to P_{read} ($L_{avi} = 2.0$)

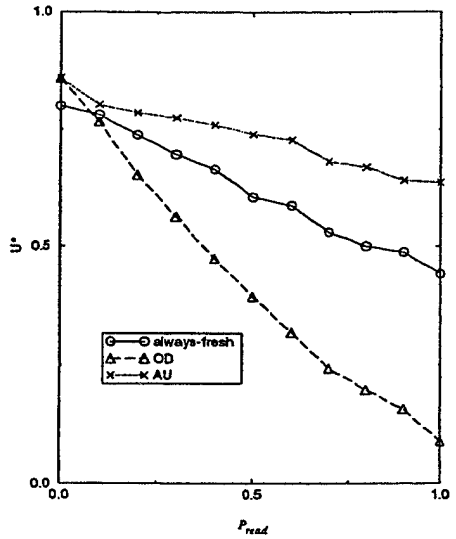


Figure 6: Breakdown utilization according to P_{read} ($L_{avi} = 5.0$)

the performance to those of absolute temporal consistency requirements.

The performance of the guarantee schemes is evaluated in terms of breakdown utilization U^* [18] over which a given transaction set is not schedulable. In our experiment, the breakdown utilization is obtained as follows: First, we generate a random transaction set that consists of 10 hard-deadline transactions. Each transaction is given a period from the uniform distribution on the interval from 100 to 2000 and its deadline is set equal to the period. The execution time c_i is given by $c_i = 0.1p_i(a_i / \sum_{j=1}^{10} a_j)$ where $a_1, \cdots, a_{10}$ are random values drawn from the uniform distribution on the interval [0,1]. This choice of the execution time is made so that the utilization of the transaction set ($U = \sum c_i/p_i$) is 0.1. Second, the generated transaction set is related to a random data set that consists of 5 real-time data objects by two parameters, P_{read} and L_{avi}. The first parameter P_{read} is the probability that a transaction τ_i read a real-time data. Thus, this parameter determines the data read set RS_{τ_i}. The second parameter L_{avi} determines the absolute validity intervals of the real-time data objects relative to the periods of the transactions in the generated transaction set. Specifically, the absolute validity interval of a real-time data object is given by multiplying the period of a randomly chosen transaction and L_{avi}. Next, the times required for updating the 5 real-time data objects are randomly drawn from the uniform distribution on the interval from 1 to 20. Finally, we multiply each c_i by a small factor δ and increase δ until the scaled up transaction set is decided to be unschedulable by a given guarantee scheme. The utilization at that point ($U^* = \sum \delta c_i/p_i$) is the breakdown utilization of the guarantee scheme for the given transaction set and data set. To observe the general effects of P_{read} and L_{avi} on the breakdown utilization, we generate 500 random samples with a given pair of P_{read} and L_{avi} and average the

breakdown utilizations for those samples.

Figure 4 compares the performance of the always-fresh, OD, and AU guarantee schemes when P_{read} varies from 0.0 to 1.0 with $L_{avi} = 0.5$. As expected, the AU guarantee scheme gives the best performance throughout since it does not suffer from needless updates. When P_{read} is low (i.e., when RS_{T_i} is small), the OD guarantee scheme outperforms the always-fresh scheme since the former needs only a small number of update requests while the latter requires a constant number of updates, which is independent of the number of real-time data objects read by each transaction. As P_{read} increases, the deadlines of transactions induced by the absolute temporal consistency are tightened because more real-time data objects are read by transactions. Thus, U^* for all the three guarantee schemes decreases. Moreover, in the OD guarantee scheme, since each instance of transactions entails update requests for all the real-time data objects read by it, as more real-time data objects are read by a transaction, a larger number of updates are required. Thus, U^* of the OD guarantee scheme decreases faster than the always-fresh scheme and there exists a cross-over point over after which the always-fresh scheme outperforms the OD guarantee scheme.

The same trends can be observed in Figures 5 and 6 where the L_{avi} value is set to 2.0 and 5.0, respectively. From these figures, we can notice that as L_{avi} increases, the U^* values for all three guarantee schemes become larger. This is because as L_{avi} increases, the deadlines of transactions get relaxed. Note that an increase in L_{avi} lengthens the absolute validity intervals of real-time data objects. In addition, as the absolute validity intervals of real-time data objects are lengthened, a smaller number of updates are required by the always-fresh guarantee scheme because the update intervals of real-time objects are also lengthened. This is also true for the AU guarantee scheme. However, in the OD guarantee scheme, the required number of updates does not change as the absolute validity intervals are lengthened since it does not depend on the absolute validity intervals but depends on the number of instances of transactions. Thus, the U^* value of the always-fresh scheme increases faster than that of the OD guarantee scheme as L_{avi} increases. This explains why the cross-over point between the two curves shifts to left in Figures 4, 5, and 6.

In summary, the AU guarantee scheme always performs best in terms of the breakdown utilization. The always-fresh guarantee scheme gives a relatively high breakdown utilization when the absolute validity intervals are large and each real-time data object is read by many transactions. In the opposite situation where the absolute validity intervals are small and each real-time data object is read by few transactions, the OD guarantee scheme outperforms the always-fresh scheme.

5 Conclusion

In this paper, we have proposed three schemes that aim to minimize the number of updates of real-time data objects that are needed to guarantee the temporal consistency of hard-deadline transactions. Among them, the AU guarantee scheme gives the best performance in terms of schedulability and spare capacity. However, the AU guarantee scheme entails aperiodic processing of updates that complicates the implementation and requires a large amount of storage to maintain all the update points of real-time data objects. On the other hand, in the OD and PU guarantee schemes all the processing including that of updates is periodic, thus simplifying the implementation, but they generate a substantial number of needless updates. Thus, there is a trade-off relationship among the three proposed schemes. We are currently developing an evaluation platform that includes a task generator, an optimization problem solver, a schedulability tester, and a simulator to study this trade-off.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1261–1269, 1989.

[3] P. S. Yu, K.-L. Wu, K.-J. Lin, and S. H. Son, "On real-time databases: Concurrency control and scheduling," *Proceedings of IEEE*, vol. 82, no. 1, pp. 140–157, Jan. 1994.

[4] R. I. Davis, "Approximate Slack Stealing Algorithms for Fixed Priority Preemptive Systems," Tech. Rep. YCS 217, Department of Computer Science, University of York, November 1993.

[5] J. P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks Fixed-Priority Preemptive Systems," in *Proceedings Real-Time Systems Symposium*, pp. 110–123, 1992.

[6] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling Slack Time in Fixed Priority Preemptive Systems," in *Proceedings Real-Time Systems Symposium*, pp. 222–231, 1993.

[7] T.-S. Tia, J. W.-S. Liu, and M. Shankar, "Aperiodic Request Scheduling in Fixed-Priority Preemptive Systems," Tech. Rep. UIUCDCS-R-94-1859, Department of Computer Science, University of Illinois, July 1994.

[8] B. Adelberg, H. Garcia-Molina, and B. Kao, "Applying Update Streams in a Soft Real-Time Database System," in *Proceedings ACM SIGMOD Conference*, pp. 245–256, 1995.

[9] B. Adelberg, B. Kao, and H. Garcia-Molina, "Database Support for Efficiently Maintaining Derived Data." to appear in Proceedings EDBT, 1996.

[10] M. Xiong, J. A. Stankovic, K. Ramamritham, D. T. Towsley, and R. Sivasankaran, "Maintaining Temporal Consistency: Issues and Algorithms," in *Proceedings of International Workshop on Real-Time Database Systems*, pp. 2–7, March 1996.

[11] K. Ramamritham, R. Sivasankaran, J. A. Stankovic, D. T. Towsley, and M. Xiong, "Integrating Temporal, Real-Time, and Active Databases," *ACM SIGMOD Record*, vol. 25, no. 1, pp. 8–12, March 1996.

[12] Y. K. Kim, *Predictability and Consistency in Real-Time Transaction Processing.* PhD thesis, Department of Computer Science, University of Virginia, May 1995.

[13] Y. K. Kim and S. H. Son, "Supporting Predictability in Real-Time Database Systems." to appear in IEEE Real-Time Technology and Applications Symposium (RTAS'96), June 1996.

[14] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *BCS Comp. J.*, vol. 29, no. 5, pp. 390–395, 1986.

[15] A. Datta, "Databases for Active Rapidly Changing data Systems (ARCS): Augmenting Real-Time Databases with Temporal and Active Characteristics." in *Proceedings of International Workshop on Real-Time Database Systems*, pp. 8–14, March 1996.

[16] B. Sprunt, *Aperiodic Task Scheduling for Real-Time Systems.* PhD thesis, Department of Computer Science. Carnegie-Mellon University, August 1990.

[17] B. Sprunt, J. P. Lehoczky, and L. Sha, "Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm," in *Proceedings Real-Time Systems Symposium*, pp. 251–258, 1988.

[18] J. P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proceedings Real-Time Systems Symposium*, pp. 166–171, 1989.

AN ADAPTIVE POLICY FOR IMPROVED TIMELINESS IN SECURE DATABASE SYSTEMS

Sang H. Son*, Rasikan David*, and Bhavani Thuraisingham†

*Department of Computer Science
University of Virginia
Charlottesville, VA 22903

†Mitre Corporation
Bedford, MA 01730

ABSTRACT

Database systems for real-time applications must satisfy timing constraints associated with transactions, in addition to maintaining data consistency. In addition to real-time requirements, security is usually required in many applications. Multilevel security requirements introduce a new dimension to transaction processing in real-time database systems. In this paper, we argue that because of the complexities involved, tradeoffs need to be made between security and timeliness. We first describe a secure two-phase locking protocol. The protocol is then modified to support an adaptive method of trading off security for timeliness, depending on the current state of the system. The performance of the Adaptive 2PL protocol is evaluated for a spectrum of security-factor values ranging from fully secure (1.0) right upto fully real-time (0.0).

Keywords: *Security, Real time, Concurrency control, Two-phase locking, Non-interference, Serializability, Scheduler, Miss percentage, Tardy time.*

This work was supported in part by ONR, by CIT, and by IBM.

1. Introduction

Database security is concerned with the ability of a database management system to enforce a security policy governing the disclosure, modification or destruction of information. Most secure database systems use an access control mechanism based on the Bell-LaPadula model [Bell 76]. This model is stated in terms of subjects and objects. An object is understood to be a data file, record or a field within a record. A subject is an active process that requests access to objects. Every object is assigned a classification and every subject a clearance. Classifications and clearances are collectively referred to as security classes (or levels) and they are partially ordered. The Bell-LaPadula model imposes the following restrictions on all data accesses:

a) *Simple Security Property*: A subject is allowed read access to an object only if the former's clearance is identical to or higher (in the partial order) than the latter's classification.
b) *The *-Property*: A subject is allowed write access to an object only if the former's clearance is identical to or lower than the latter's classification.

The above two restrictions are intended to ensure that there is no flow of information from objects at a higher access class to subjects at a lower access class. Since the above restrictions are mandatory and enforced automatically, the system checks security classes of all reads and writes. Database systems that support the Bell-LaPadula properties are called multilevel secure database systems (MLS/DBMS).

The Bell-LaPadula model prevents direct flow of information from a higher access class to a lower access class, but the conditions are not sufficient to ensure that security is not violated indirectly through what are known as covert channels [Lamp 73]. A covert channel allows indirect transfer of information from a subject at a higher access class to a subject at a lower access class. An important class of covert channels that are usually associated with concurrency control mechanisms are timing channels. A timing channel arises when a resource or object in the database is shared between subjects with different access classes. The two subjects can cooperate with each other to transfer information.

A real time database management system (RTDBMS) is a transaction processing system where transactions have explicit timing constraints. Typically a timing constraint is expressed in the form of a deadline, a certain time in the future by which a transaction needs to be completed. In a real-time system, transactions must be scheduled and processed in such a way that they can be completed before their corresponding deadline expires. Conventional data models and databases are not adequate for time-critical applications. They are designed to provide good average performance, while possibly yielding unacceptable worst-case response times. As advances in multilevel security take place, MLS/DBMSs are also required to support real-time requirements. As more and more of such systems are in use, one cannot avoid the need for integrating real-time data processing techniques into MLS/DBMSs. In [Son 93], the security impact on real-time database systems is studied, but to the best of our knowledge, no work has been reported on developing DBMSs which are multilevel secure and which support real-time applications.

Concurrency control is used in databases to manage the concurrent execution of operations by different subjects on the same data object such that consistency is maintained. In multilevel secure databases, there is the additional problem of maintaining consistency without introducing covert channels. For a more detailed description of and a possible solution to the problem of concurrency control in secure databases, the reader is referred to [Dav 93]. In this paper, we discuss the additional issues that arise when transactions in a secure database have timing constraints associated with them. Section 2 is an overview of related work in secure concurrency control. In Section 3, the problems associated with time-constrained secure concurrency control are studied and possible solutions are evaluated. In Section 4, a specification of the

secure two phase locking protocol [Dav 93] is provided. The inherent limitations of a fully secure concurrency control protocol for a secure database and a method by which security can be partially compromised for improved deadline cognizance is presented in Section 5. In Section 6, a simulation system for a secure database system is described and the performance of a modified Secure 2PL for various degrees of maintenance of the security properties is evaluated. Section 7 concludes the paper.

2. Background

2.1 Correctness Criteria for Secure Schedulers

Covert channel analysis and removal is the single most important issue in multilevel secure concurrency control. The notion of *non-interference* has been proposed [Gogu 82] as a simple and intuitively satisfying definition of what it means for a system to be secure. The property of *non-interference* states that the output as seen by a subject must be unaffected by the inputs of another subject at a higher access class. This means that a subject at a lower access class should not be able to distinguish between the outputs from the system in response to an input sequence including actions from a higher level subject and an input sequence in which all inputs at a higher access class have been removed [Keef 90a].

An extensive analysis of the possible covert channels in a secure concurrency control mechanism and the necessary and sufficient conditions for a secure, interference-free scheduler are given in [Keef 90a]. Three of these properties are of relevance to the secure two phase locking protocol discussed in this paper. For the following definitions, given a schedule s and an access level l, *purge(s,l)* is the schedule with all actions at a level $> l$ removed from s.

1) *Value Security*: A scheduler satisfies this property if values read by a subject are not affected by actions with higher subject classification levels. Stated formally, for an input schedule p, the output schedule s is said to be *value secure* if *purge(s,l)* is view equivalent to the output schedule produced for *purge(p,l)*.

2) *Delay Security*: This property ensures that the delay experienced by an action is not affected by the actions of a subject at a higher classification level. An action a_1 is said to be delayed with respect to another action a_2 if and only if:

- The action a_1 appears before a_2 in the input schedule.
- The action a_1 follows a_2 in the output schedule.
- The action a_2 is the last action in the input schedule for which the first two conditions are satisfied.

For an input schedule p and an output schedule s, a scheduler is *delay secure* if for all levels l in p, each of the actions a_1 in *purge(p,l)* is delayed with respect to a_2 in the output schedule produced for *purge(p,l)* if and only if it is delayed with respect to a_2 in *purge(s,l)*.

3) *Recovery Security*: A set of transactions is in a deadlock state when every transaction in the set is waiting for an event that can only be caused by another transaction in the set (such as release of a lock). Deadlock is a problem unique to locking protocols and is not an issue in timestamp schedulers and optimistic concurrency control. Even these schedulers, however, can reach a state from which they cannot continue without aborting one or more transactions. For simplicity, these two conditions are lumped together and called as deadlock [Keef 90a].

When a deadlock is detected, some of the actions in the schedule must be aborted, allowing the others to proceed. If resolving the deadlock can allow a high-level transaction to modify the values read by a low

level transaction or to affect the delay a low level transaction experiences, a covert channel can arise. When a deadlock occurs, other channels are available for signaling in addition to those protected by *value security* and *delay security*. The following condition takes care of these channels:

A scheduler is *recovery secure* for all schedules p if, on the arrival of an action A_X for scheduling:

1) If a deadlock occurs, resulting in a set of actions D being rolled back, then for all subject classification levels l in p, which dominate one of those in D, a deadlock also occurs in response to the schedule *purge(l,p)* on the arrival of the action A_X, with the actions *purge(l,D)* being rolled back.
2) If no deadlock occurs on the arrival of A_X, then for all subject classification levels l in p, it does not occur on the arrival of A_X in the input schedule *purge(l,p)*.

Recovery security ensures that the occurrence of a deadlock appears the same to a low-level subject, independent of whether higher level actions are in the schedule or not. The actions taken to recover from deadlock are also not affected by the presence of higher level transactions.

2.2 Approaches to Secure Concurrency Control

In this section, we discuss some of the existing concurrency control algorithms and see why they yield unacceptable performance for secure databases.

2.2.1. Locking and Timestamp Ordering

Locking will fail in a secure database because the security properties prevent actions in a transaction T_1 at a higher access class from delaying actions in a transaction T_2 at a lower access class (e.g. when T_2 requests a conflicting lock on a data item on which T_1 holds a lock). Timestamp ordering fails for similar reasons, with timestamps taking the role of locks, since a transaction at a higher access class cannot cause the aborting of another transaction at a lower access class. There are approaches for adapting locking and timestamping techniques for MLS/DBMSs. For example, in [Rubi 92], two copies of a data item are maintained at level L, one for subjects at level L and one for subjects which dominate level L. Subsequently, locking and timestamp based concurrency control techniques are proposed.

2.2.2. Optimistic Concurrency Control

Optimistic concurrency control for a secure database can be made to work by ensuring that whenever a conflict is detected between a transaction T_h at a higher access class in its validation phase and a transaction T_l at a lower access class, the transaction at the higher access class is aborted, while the transaction at the lower access class is not affected. A major problem with using optimistic concurrency control is the possible *starvation* of higher-level transactions. For example, consider a long-running transaction T_h that must read several lower-level data items before the validation stage. In this case, there is a high probability of conflict and as a result, T_h may have to be rolled back and restarted an indefinite number of times. This issue will be analyzed in greater detail in Section 6.

2.2.3. Multiversion Timestamp Ordering

A secure version of the MVTO scheduler is presented in [Keef 90b]. The difference between Basic MVTO and Secure MVTO is that Secure MVTO will sometimes assign a new transaction a timestamp that is earlier than the current timestamp. This effectively moves the transaction into the past with respect to active transactions. To be more precise, when a transaction begins, it is assigned a timestamp that precedes

the timestamps of all transactions active at strictly dominated access classes and that follows the timestamps of all transactions at its own access class. This approach to timestamp assignment is what makes it impossible for a transaction to invalidate a read from a higher access class.This method has the drawback that transactions at a higher access class are forced to read arbitrarily old values from the database because of the timestamp assignment. This problem can be especially serious if most of the lower level transactions are long running transactions.

2.2.4. Replicated Database Architectures

In [Koga 90], Kogan and Jajodia discuss a concurrency control mechanism in secure databases using a replicated architecture which guarantees serializable execution of concurrent transactions. This protocol, however, has the same problem as that associated with the Secure MVTO algorithm, where transactions at a higher access class might be forced to read arbitrarily old values.

3. Security and Real-Time Requirements

There are two approaches which can be explored to design real-time concurrency control algorithms for databases:

- Improvements to existing concurrency control algorithms for secure databases to make them time-cognizant.
- Correctness criteria weaker than serializability which would result in increased concurrency. A discussion of this approach, however, is beyond the scope of this paper.

There have been a number of papers in the real-time databases literature that have explored these approaches with respect to conventional databases [Abbo 92],[Sha 87],[Hari 90], [Son 92]. The problem arises when these approaches are applied to secure databases, because covert channels can be introduced by priority based scheduling. All existing real-time systems schedule transactions based on some priority scheme. The priority usually reflects how close the transaction is to missing its deadline. Priority-based scheduling of real-time transactions, however, interacts with the property of non-interference which has to be satisfied by secure schedulers [Keef 90a]. Take the sequence of transactions input to a scheduler as shown:

T_1 (SECRET)	:	*R(X)*			
T_2 (UNCLASSIFIED)	:		*W(X)*		
T_3 (UNCLASSIFIED)	:			*W(X)*	
T_4 (UNCLASSIFIED)	:				*R(X)*

Assume that T_1, T_2 and T_3 have priorities *5*, *7* and *10* respectively and the priority assignment scheme is such that if *priority*(T_2) > *priority*(T_1), then T_2 has greater criticalness and has to be scheduled ahead of T_1. In the above example, T_2 and T_3 are initially blocked by T_1 when they arrive. When T_1 completes execution, T_3 is scheduled ahead of T_2, since it has a greater priority than T_2 and the transaction execution order would be T_1 T_3 T_2 T_4. However, if the transaction T_1 is removed, the execution order would be T_2 T_3 T_4 because T_2 would have been scheduled as soon as it had arrived. The presence of the SECRET transaction T_1 thus changes the value read by the UNCLASSIFIED transaction T_4, which is a violation of *value security*. For the same reason *delay security* is also violated, because the presence of T_1 delays T_2 with respect to T_3.

It is easy to see that the problems with any concurrency control mechanism are present because a higher

level transaction cannot interfere with the execution of a lower level transaction. This makes the mechanism unfair, because higher level transactions are starved for service.

A transaction at a higher access class:

- Cannot cause the aborting of a transaction at a lower access class. If it is allowed to do so, it is possible that it can control the number of times a lower level transaction is aborted, thereby opening a covert channel.
- Cannot conflict with a transaction at a lower access class. If such a conflict does occur, the higher level transaction has to be blocked or aborted, not the low level transaction.
- Cannot be granted greater priority of execution over a transaction at a lower access class.

A real-time secure concurrency control algorithm must possess two characteristics - speed and minimal deadline miss percentage. The secure two phase locking protocol [Dav 93] was shown to yield best average case performance among all the secure concurrency control approaches whose performance was evaluated in [Son 94]. We therefore use it as a basis for our solution to the problem of real-time secure concurrency control. From our discussion earlier in this section, it is clear that priority-based transaction scheduling is not feasible for a fully secure database system. Therefore, for minimizing deadline miss percentage, we take the approach that partial security violations under certain conditions are permissible, if it results in substantial gain in time cognizance. In the next section, the Secure 2PL protocol is described, and in the subsequent section, the design of a tradeoff factor between the security properties and the real-time requirements is explained.

4. Secure Two-Phase Locking

Basic two-phase locking does not work for secure databases because a transaction at a lower access class (say T_l) cannot be blocked because of a conflicting lock held by a transaction at a higher access class (T_h). If T_l were somehow allowed to continue with its execution in spite of the conflict, then non-interference would be satisfied. The basic principle behind the secure two-phase locking protocol is to try to simulate execution of Basic 2PL without blocking of lower access class transactions by higher access class transactions.

4.1. An Example

Consider the two transactions in example 1:

T_1 (SECRET) : $r[x]$ ••• c_1
T_2 (UNCLASSIFIED) : $w[x]$ c_2

EXAMPLE 1

Basic two phase locking would fail because $w_2[x]$ would be blocked waiting for T_1 to commit and release $rl_1[x]$. In our modification to the two-phase locking protocol, T_2 is allowed to set a virtual lock $vwl_2[x]$, write onto a version of x local to T_2 and continue with the execution of its next operation, i.e. c_2. When T_1 commits and releases the lock on x, T_2's virtual write lock is upgraded to a real lock and $w_2[x]$ is performed. Until $w_2[x]$ is performed, no conflicting action is allowed to set a lock on x. The sequence of operations performed is therefore, $rl_1[x]\ r_1[x]\ vwl_2[x]\ c_2$ ••• $c_1\ ru_1[x]\ wl_2[x]\ w_2[x]\ wu_2[x]$.

This modification alone is not enough, as illustrated in Example 2:

T_1 (SECRET)	:	$r[x]$		$r[y]$	c_1
T_2 (UNCLASSIFIED)	:		$w[x]$ $w[y]$		c_2

EXAMPLE 2

The sequence of operations that would be performed is $rl_1[x]$ $r_1[x]$ $vwl_2[x]$ $vw_2[x]$ $wl_2[y]$ $w_2[y]$ c_2. After these operations, deadlock would occur because $r_1[y]$ waits for $w_2[y]$ to release its virtual lock and $vw_2[x]$ waits for $r_1[x]$ to release its lock. This deadlock would not have occurred in basic two-phase locking. Note that our aim of trying to simulate execution of basic two phase locking is not being achieved. On closer inspection, it is obvious that this problem arises because $w_2[y]$ is allowed to proceed with its execution even though $w_2[x]$ could only write onto a local version of x because of the read lock $rl_1[x]$ set by T_1. To avoid this problem, for each transaction T_i, two lists are maintained - *before*(T_i) which is the list of active transactions that precede T_i in the serialization order and *after*(T_i) which is the list of active transactions that follow T_i in the serialization order. This idea is adapted from [Son 92], where *before_cnt* and *after_cnt* are used to dynamically adjust the serialization order of transactions. The following additions are made to the basic two-phase locking protocol:

1) When an action $p_i[x]$ sets a virtual lock on x because of a real lock $ql_j[x]$ held by T_j, then T_i and all transactions in *after*(T_i) are added to *after*(T_j), T_j and all transactions in *before*(T_j) are added to *before*(T_i).
2) When an action $w_i[x]$ arrives and finds that a previous action $w_i[y]$ (for some data item y) has already set a virtual write lock $vwl_i[y]$, then a dependent lock $dvwl_i[x]$ is set with respect to $vwl_i[y]$.
3) When an action $p_i[x]$ arrives and finds that a conflicting virtual or dependent lock $vql_j[x]$ or $dvql_j[x]$ has been set by a transaction T_j which is in *after*(T_i), then $p_i[x]$ is allowed to set a lock on x and perform $p_i[x]$ in spite of the conflicting lock.
4) A dependent virtual lock $dvp_i[x]$, dependent on some action $q_i[y]$ is upgraded to a virtual lock when $vql_i[x]$ is upgraded to a real lock.

The maintenance of a serialization order and the presence of dependent locks are necessary to prevent uncontrolled acquisition of virtual locks by transactions at lower access classes.

For example 2, the sequence of operations that would now be performed is $rl_1[x]$ $r_1[x]$ $vwl_2[x]$ $dvwl_2[y]$ c_2 $rl_1[y]$ $r_1[y]$ c_1 $ru_1[x]$ $ru_1[y]$ $wl_2[x]$ $w_2[x]$ $wu_2[x]$ $wl_2[y]$ $w_2[y]$ $wu_2[y]$.

The conditions given above are necessary to ensure that Delay Security is satisfied, but Value Security could still be violated as shown in Example 3.

T_1 (TOP SECRET)	:	$r_1[y]$	$r_1[x]$	•••		c_1
T_2 (SECRET)	:				$r_2[z]$	c_2
T_3 (CLASSIFIED)	:	$w_3[x]$		$w_3[z]$		c_3
T_4 (UNCLASSIFIED)	:		$w_4[y]$	$w_4[z]$		c_4

EXAMPLE 3

When T_4 requests a write lock on 'y', T_1 already holds a real lock on 'y', i.e., T_1 is set before T_4 in the serialization order. When T_1 requests a read lock on 'x', T_3 already holds a write lock on 'x', i.e., T_3 is set before T_1 in the serialization order. When T_3 requests a write lock on 'z', T_4 already holds a dependent lock on 'z'. However, T_3 is before T_1 in the serialization order and T_1 is, in turn, before T_4 in the serialization order, i.e., T_3 is before T_4 in the serialization order and so, $w_3[z]$ is allowed to supersede $w_4[z]$ in the lock queue for 'z'. Now, when $r_2[z]$ is submitted, it will read the value of 'z' written by T_4. However, if T_1 had not been present, then T_3 would not have been before T_4 in the serialization order. This means that

$w_3[z]$ would not have superseded $w_4[z]$ in the lock queue for 'z', i.e., $r_2[z]$ would have read the value of 'z' written by T_3. Since the value read by a SECRET transaction is affected by the presence of a transaction at the TOP SECRET level, Value Security is being violated.

In general, an action in transaction T_1 can supersede an action in transaction T_2 if T_1 is before T_2 in the serialization order. Now, Value Security could be violated if T_1 is before T_2 in the serialization order because of the presence of a transaction T_3 at a higher access class, i.e., if T_3 is between T_1 and T_2 in the serialization order. To overcome this problem, the action in T_1 is not allowed to supersede the action in T_2 if such a case arises.

4.2. The Three Lock Types

The semantics of the three different kinds of locks are explained below:

1) *Real Lock (of the form $pl_i[x]$):* A real lock is set for an action $p_i[x]$ if no other conflicting action has a real lock or a virtual lock on x. The semantics of this lock are identical to that of the lock in basic two-phase locking.

2) *Virtual Lock (of the form $vpl_i[x]$):* A virtual lock $vpl_i[x]$ is set for an action $p_i[x]$ if a transaction at a higher access class holds a conflicting lock on x ($p_i[x]$ has to be a write to satisfy the Bell-LaPadula properties). The virtual lock is non-blocking. Once a virtual lock $vpl_i[x]$ is set, $p_i[x]$ is added to *queue[x]* and the next action in T_i is ready for scheduling. When $p_i[x]$ gets to the front of the lock queue, its virtual lock is upgraded to a real lock and $p_i[x]$ is submitted to the scheduler. A virtual lock holding action $vpl_i[x]$ can be superseded in the lock queue by a conflicting action $q_j[x]$ if T_j is in *before(T_i)*.

3) *Dependent Virtual Lock (of the form $dvpli[x]$):* A dependent virtual lock is set for an action $p_i[x]$ (where p is a write) if a previous write $w_i[y]$ in the same transaction holds a virtual lock. An action $p_i[x]$ which holds a dependent virtual lock with respect to another action $w_i[y]$ is not allowed to set a real lock or a virtual lock unless $w_i[y]$'s virtual lock is upgraded to a real lock. The dependent lock is non-blocking and can be superseded by a conflicting action $q_j[x]$ if T_j is before T_i in the serialization order.

4.3. The Secure Two-Phase Locking Protocol

The Secure 2PL scheduler manages its locks according to the following rules:

When an action $p_i[x]$ is submitted, one of three possible cases can arise:

Case 1 $(p = read) \wedge \exists (wLock(T_i, x) \vee VwLock(T_i, x) \vee DVwLock(T_i, x))$
Read value of x written by $w_i[x]$;

Case 2 $((p = write) \wedge \exists rLock(T_i, x))$
Upgrade $rLock(T_i,x)$ to $wLock(T_i,x)$;

Case 3
$$\exists j\,((qLock(T_j, x) \vee VqLock(T_j, x) \vee DVqLock(T_j, x) \vee Wait(T_j, x)) \wedge (j \neq i) \wedge ((p = write) \vee (q = write)))$$

```
pos := length(queue[x]);
while (pos > 0)
        T_j ← transaction at queue[x] → pos;
        if (operation(queue[x] → pos) conflicts with p)
                if (T_i ∈ before(T_j))
                        ∀T_3((T_3 ∈ before(Tj)) ∧ (T_3 ∈ after(T_i))
                            ∧ (SL(T_3) > SL(T_i)) ∧ (SL(T_3) > SL(T_j)) )
                                abort T_3; /* value security violation */
                        if (lock(queue[x] → pos) is a real lock)   /* deadlock */
                                call victim selection routine;
                                if (victim = T_i)
                                        exit;
                                else
                                        continue;
                if (T_i ∉ before(T_j) )
                        if( (p = write) ∧ (∃z, VwLock(T_i, z)) )
                                Insert(queue,DVwLock(T_i,x),pos); /* insert after pos */
                                dependent(w_i[z]) := w_i[x];
                        else if (SL(T_i) < SL(T_j))
                                Insert(queue,VwLock(T_i,x),pos);
                        else
                                Insert(queue,pWait(T_i,x),pos);
                        before(T_i) := before(T_i) ∪ {T_j} ∪ before(T_j) ;
                        after(T_j) := after(T_j) ∪ {T_i} ∪ after(T_i)
        pos := pos - 1;
if (pos = 0)
        Insert(queue[x],pLock(T_i,x),pos);
```

The cases above arise when an action is submitted. When a lock on a data item x is released and the action at the front of *queue[x]* does not have a dependent lock, $pl_i[x]$ is set and $p_i[x]$ submitted. If $p_i[x]$ had held a virtual lock, all locks dependent on it are upgraded to real or virtual locks depending on whether they were at the front of their respective lock queues or not.

In addition, the following two conditions are to be satisfied by the transaction manager:

1) A transaction is allowed to commit, even if virtual writes performed by the transaction have not been committed to stable storage, i.e the writes are still on some queue[x] waiting to set a real lock on x.

2) Once an action $p_i[x]$ acquires a lock on a data item, the lock is not released until T_i commits and $p_i[x]$ is performed on the data item x in stable storage. Therefore, in most cases, all the locks that a transaction holds are not released when it commits.

4.4 Deadlock Detection and Resolution

The secure two-phase locking protocol is not free from deadlocks. For example, consider the input schedule:

T_1 (SECRET)	:	$r[x]$		$r[y]$
T_2 (UNCLASSIFIED)	:		$w[y]$		$w[x]$

The sequence of operations performed are $rl_1[x]\ r_1[x]\ wl_2[y]\ w_2[y]\ vwl_2[x]\ c_2$. After these operations, deadlock would occur because $r_1[y]$ waits for $w_2[y]$ to release its lock and $vw_2[x]$ waits for $r_1[x]$ to release its lock. Deadlocks can be detected by constructing a wait-for graph [Bern 87]. A scheduler recovers from a deadlock by aborting one of the transactions in the cycle. A secure scheduler must ensure that Recovery Security is not violated by the choice of a victim. This can be guaranteed by aborting a transaction at the highest access class in the cycle detected in the wait-for graph.

5. An Adaptive Security Policy

A detailed performance analysis of Secure 2PL is provided in [Son 94]. The conclusion was that it exhibits a much better response time characteristic than Secure OCC. Its operating region (the portion of the curve before the saturation point) is much larger than that of Secure OCC (Figure 1). Further, staleness is not an issue in Secure 2PL as with Secure MVTO. However, this alone does not suffice when timing constraints are present on transactions. In Secure 2PL, transaction scheduling order is determined purely by the order in which transactions acquire locks. No conscious effort is made to schedule transactions according to their priority, or according to how close a transaction is to meeting its deadline. In a real-time database system this is unacceptable. In Section 3, we have seen that priority-driven scheduling of transactions could lead to security violations. It is our claim that the security properties have to be sacrificed to some extent to ensure a certain degree of deadline cognizance.

An introduction to the problem of covert channels was given in Section 2. A covert timing channel is opened between two collaborating transactions - one at a higher access class and the other at a lower access class - if the higher access class transaction can influence the delay seen by a lower access class transaction. The *bandwidth* of a covert channel is a measure of how easy it is for the higher access class transaction to control the delay seen by the lower access class transaction. If there is a great degree of randomness in the system, i.e., an indeterminate number of transactions could be affecting the delay that the higher access class transactions wants a lower access class transaction to experience, then the bandwidth is low. On the other hand, if the higher access class transaction knows that the lower access class transaction to which it wants to transmit information is the only other transaction in the system, then the bandwidth is infinite. Therefore, when security has to be sacrificed, a policy that keeps the bandwidth of the resulting covert channel to a minimum is desirable. To ensure this, the security policy has to be adaptive, i.e., determining whether security is to be violated or not when a conflict arises should depend on the current state of the system and not on a static, predecided property.

Our adaptive policy to resolve conflicts between lock holding and lock requesting transactions is based on past execution history. Whenever a transaction T_1 requests a lock on a data item x on which another transaction T_2 holds a conflicting lock, there are two possible options:

- T_1 could be blocked until T_2 releases the lock.
- T_2 could be aborted and the lock granted to T_1.

If T_1 were at a higher security level than T_2, the latter option would be a violation of security. However, if T_1 has greater priority than T_2, then the latter option would be the option taken by a real-time concurrency control approach. In our approach, we strike a balance between these two conflicting options by looking up past history. A measure of the degree to which security has been violated in the past is calculated. A similar measure of the degree to which the real-time constraints have not been satisfied can be obtained from the number of deadlines missed in the past. These two measures are compared and depending on which value is greater, either the security properties are satisfied or the higher priority transaction is given the right to execute.

The two factors that are used to resolve a conflict are:

- Security Factor (SF): (number of conflicts for which security is maintained/ total number of conflicts) * Difference in security level between the two conflicting transactions.

- Deadline Miss Factor (DMF): number of transactions that missed their deadline/Total number of transactions committed

Two factors are involved in the calculation of SF. The first factor is the degree to which security has been satisfied in the past, measured by the number of conflicts for which security has been maintained. Secondly, we also assume that the greater the difference in security levels between the transactions involved in the conflict, the more important it is to maintain security. DMF is determined only by the number of deadline misses in the past. Note that for a comparison with DMF, (1 - SF) has to be used, since (1 - SF) is a measure of the degree to which security has been violated. Now, a simple comparison (1 - SF) > DMF is not enough, since different systems need to maintain different levels of security. Therefore, we define two weighting factors, α and β for (1 - SF) and DMF respectively. If $\alpha * (1 - SF) > \beta * DMF$, then for the conflict under consideration, the security properties are more important and therefore the conflict is decided in favor of the transaction at a lower access class. If the opposite is true, then the transaction with higher priority is given precedence. Note that at low conflict rates, it is possible to satisfy both the security and the real-time requirements simultaneously. As a result the comparison is not made until the DMF reaches a certain threshold value DMISS_THRESH. The parameters DMISS_THRESH, α and β can be tuned for the desired level of security. A very high value of DMISS_THRESH or a very high value of α compared to β would result in SF being maintained at 1.0, i.e., for all conflicts the security properties are satisfied. A very high value of β compared to α would result in an SF value of 0.0, i.e., the behavior would be identical to that of 2PL-HP [Abbo 92]. For a desired value of SF between 0 and 1, the values of α, β and DMISS_THRESH would have to be tuned based on the arrival rate of transactions.

The hybrid protocol is defined by the following rules:

If a conflict between a lock holding transaction T_1 and a lock requesting transaction T_2 arises, the conflict is settled using the following rules:

- *If DMF < DMISS_THRESH then follow the steps taken by the Secure 2PL protocol*
- *Else If $\alpha * (1 - SF) > \beta * DMF$, follow the steps taken by the Secure 2PL protocol*
- *Else break the conflict in favor of the transaction with the higher priority*

6. Performance Evaluation

In this section, we present the results of our performance study of the Adaptive Secure 2PL protocol for a wide range of values of SF. The two main goals of our performance analysis are:

- To determine miss percentages for varying transaction arrival rates for various values of SF.
- Since our model assumes a soft deadline system, the second factor that has been measured is tardy time - the difference between the commitment time and deadline for late transactions.

6.1 Simulation Model

Central to the simulation model is a single-site disk resident database system operating on shared-memory multiprocessors [Lee 93]. The system consists of a disk-based database and a main memory cache. The unit of database granularity is the *page*. When a transaction needs to perform an operation on a data item it accesses a page. If the page is not found in the cache, it is read from disk. CPU or disk access is through an M/M/k queueing system, consisting of a single queue with *'k'* servers (where *'k'* is the number of disks or CPUs). The service times for CPU operations and disk I/O are specified as model parameters in Table 1. Since we are concerned only with providing security at the concurrency control level, the issue of providing security at the operating system or resource scheduling layer is not considered in this paper. That is the reason why we do not consider a secure CPU/disk scheduling approach. Our assumption is that the lower layers provide the higher concurrency control layer with a fair resource scheduling policy.

In the model for Adaptive Secure 2PL, the execution of a transaction consists of multiple instances of alternating data access requests and data operation steps, until all the data operations in it complete or it is aborted. When a transaction makes a data request, i.e., lock request on a data object, the request must go through concurrency control to obtain a lock on the data object. If $\alpha * (1 - SF) < \beta * DMF$, then if the transaction's priority is greater than all of the lock holders, the holders are aborted and the transaction is granted a lock; if the transaction's priority is lower, it waits for the lock holders to release the lock [Abbo 92]. However, if $\alpha * (1 - SF) > \beta * DMF$, then the steps taken by the Secure 2PL are followed. If the request for a lock is granted, the transaction proceeds to perform the data operation, which consists of a possible disk access (if the data item is not present in the cache) followed by CPU computation. However, if only a virtual or dependent lock is granted, the transaction only does CPU computation, since the operation should only be performed on a local version. If the request for the lock is denied (the transaction is blocked), the transaction is placed into the data queue. When the waiting transaction is granted a lock, only then can it perform its data operation. Also, when a virtual lock for an operation is upgraded to a real lock, the data operation requires disk access and CPU computation. At any stage, if a deadlock is detected, the transaction to be aborted to break the deadlock is determined, aborted and restarted. When all the operations in a transaction are completed, the transaction commits. Even if a transaction misses its deadline, it is allowed to execute until all its actions are completed.

6.2 Parameters and Performance Metrics

Table 1 gives the names and meanings of the parameters that control system resources. The parameters, *CPUTime* and *DiskTime* capture the CPU and disk processing times per data page. Our simulation system does not explicitly account for the time needed for data operation scheduling. We assume that these costs are included in *CPUTime* on a per-data-object basis. The use of a database cache is simulated using probability. When a transaction attempts to read a data page, the system determines whether the page is in cache or disk using the probability *BufProb*. If the page is determined to be in cache, the transaction can continue processing without disk access. Otherwise disk access is needed.

Table 2 summarizes the key parameters that characterize system workload and transactions. Transac-

tions arrive in a Poisson stream, i.e., their inter-arrival rates are exponentially distributed. The *ArriRate* parameter specifies the mean rate of transaction arrivals. The number of data objects accessed by a transaction is determined by a normal distribution with mean *TranSize*, and the actual data objects to be accessed are determined uniformly from the database.

The assignment of deadlines to transactions is controlled by the parameters *MinSlack* and *MaxSlack*, which set a lower and upper bound, respectively, on a transaction's slack time. We use the formula for deadline-assignment to a transaction.

Deadline = AT + Uniform(*MinSlack,MaxSlack*) * *ET*

AT and ET denoting the arrival time and execution time, respectively. The execution time of a transaction used in this formula is not an actual execution time, but a time estimated using the values of parameters *TranSize, CPUTime* and *DiskTime*. The priorities of transactions are decided by the *Earliest Deadline First* policy.

The primary performance metric used is miss percentage, which is the ratio of the number of transactions that do not meet their deadline to the total number of transactions committed. The second performance metric is tardy time.

6.3 Experiments and Results

An event-based simulation framework was written in 'C'. For each experiment, we ran the simulation with the same parameters for 6 different random number seeds. Each simulation run was continued until 200 transactions at each access class were committed. For each run, the statistics gathered during the first few seconds were discarded in order to let the system stabilize after an initial transient condition. For each experiment the required performance measure was measured over a wide range of workload. All the data reported in this paper have 90% confidence intervals, whose endpoints are within 10% of the point estimate.

6.3.1 Experiment 1: Secure 2PL vs Secure OCC vs Secure MVTO

The first experiment was conducted to prove that the average case performance of Secure 2PL is better than that of the other existing secure concurrency control algorithms. Note that good average case performance is also essential for a real-time system. A slow system cannot be expected to meet timing constraints. In this experiment, the response times of Secure 2PL, Secure OCC and Secure MVTO at each security level were measured for varying arrival rates. The resulting graphs are shown in Figures 1 and 2. At low arrival rates, the response times are more or less the same for all three approaches. This is because the contention levels are low and majority of time is spent in disk access and CPU access rather than in resource queues, lock queues or transaction aborts. As the arrival rate increases the impact of these factors increases, and depending on how much they increase in each concurrency control approach, the performance varies. In Secure OCC, the saturation point is reached much earlier than in Secure 2PL and Secure MVTO. As the arrival rate increases, the contention level for data items increases. As a result, when a higher access class transaction reaches its validation stage, invariably it would have conflicted with a transaction at a lower access class and would therefore have to be aborted and restarted. It was found that the system reached a stage where, at the highest access class, transactions were repeatedly being aborted and restarted, resulting in the steep increase in response time at the saturation point. Now, since transactions are not being committed while arrival rate is unchanged, the net number of active transactions keeps increasing, increasing contention for the finite resources. It is because of this phenomenon that the

Table 1: System Resource Parameters

Parameter	Meaning	Base Value
DBSize	Number of data pages in database	350
NumCPUs	Number of processors	2
NumDisks	Number of disks	4
CPUTime	CPU time for processing an action	15 msec
DiskTime	Disk service time for an action	25 msec
BufProb	Prob. of a page in memory buffer	0.5
NumSecLevels	Num. of security levels supported	6

Table 2: Workload Parameters

Parameter	Meaning	Base Value
ArriRate	Mean transaction arrival rate	-
TranSize	Average Transaction Size	6
RestartDelay	Mean overhead in restarting	1 msec
MinSlack	Minimum slack factor	2
MaxSlack	Maximum slack factor	8

response time for lower access class transactions increases also at the saturation point. The performance margin of Secure 2PL at level 0 over Secure 2PL at level 5 is small, indicating a greater measure of fairness than in Secure OCC. In addition, the saturation point is reached at a much higher arrival rate in Secure 2PL than in Secure OCC. Of course, the response time graph for Secure MVTO is the best, since no transactions are blocked or aborted. The performance margin of Secure MVTO at level 0 over Secure MVTO at level 5 is negligibly small. The rate of increase in response time after the saturation point is also much less than that of both Secure OCC and Secure 2PL. However, in Secure MVTO, the good response time is offset by an unacceptable *staleness* factor. This is all the more critical in a real-time environment, where data could have temporal constraints associated with them.

6.3.2 Experiment 2: Adaptive Secure 2PL - Miss Percentage

In this experiment, the miss percentages for Adaptive Secure 2PL are measured for three different settings of the SF values - fully secure (SF = 1.0), no security (SF = 0.0) and partially secure (SF = 0.5). The resulting graph is shown in Figure 3. Since we are considering a real-time database system, we restrict attention to the portion of the graph where miss percentages are less than 10%. The performance after the saturation point is not an issue. As is to be expected, 2PL-HP has the lowest miss percentage, with the curve for 2PL (SF=0.5) falling between 2PL-HP and Secure 2PL. However, one does not see a progressive decrease in miss percentage from SF=1.0 to SF=0.0. The miss percentage increases from SF=1.0 to SF=0.65, but subsequently decreases at SF=0.5 up to SF=0.0. This is because at higher values of SF (0.65 to 1.0), the number of actions scheduled according to priority is very low. However, since the system is not fully secure, some transactions are being aborted, when security needs to be violated by the Adaptive 2PL protocol. The problem is that the former factor does not offset the latter factor, resulting in increased miss percentage. As SF decreases, the former factor increases, resulting in more transactions meeting their deadline.

6.3.3 Experiment 3: Adaptive Secure 2PL - Tardy Time

In this experiment, the average tardy times for Adaptive Secure 2PL are measured for three different settings of the SF values - fully secure (SF = 1.0), no security (SF = 0.0) and partially secure (SF = 0.5). The resulting graph is shown in Figure 4. The results obtained conform with the results obtained for the miss percentage calculation. The tardy time is maximum for Secure 2PL, with the curve for Adaptive 2PL (SF = 0.5) falling between that of Secure 2PL and 2PL-HP.

7. Conclusions

In this paper, we have presented an approach to scheduling transactions to meet their timing constraints in a secure database. Our simulation results substantiate our claim that an adaptive security policy that sacrifices the security properties to some extent can significantly improve the deadline miss performance. The work described in this paper is more a direction for future research than a concrete solution to the problem of secure real-time concurrency control. There are a number of issues that need to be looked into. First of all, a proper characterization of the bandwidth of a covert channel that can arise given a particular value of SF needs to be derived. Applications might express a desired level of security in terms of a maximum admissible bandwidth of a potential covert channel. Unless there is a way of determining to what extent a security policy satisfies the security properties, one cannot determine whether the policy is suitable for the application or not. Secondly, in this paper we have considered a simple tradeoff between deadline miss percentage and security. A tradeoff could also have been made between alternative factors depending on the application. Thirdly, we have restricted ourselves to a soft deadline system with no overload management policy. It would be interesting to see how a policy to screen out transactions that are about to miss

their deadline would affect performance. Finally, in this paper, we have restricted ourselves to the problem of real-time secure concurrency control in a database system. Some of the other issues that need to be considered in designing a comprehensive real-time multilevel secure database system (MLS/RTDBMS) are dealt with in [Son 93]. Various types of MLS/RTDBMSs need to be identified and architectures and algorithms developed for each type of system. Tradeoffs need to be made between security, timeliness and consistency on a case-by-case basis.

References

[Abbo 92] Robert K. Abbott and Hector Garcia-Molina. "Scheduling Real-Time Transactions: A Performance Evaluation", ACM Transactions on Database Systems, Vol. 17, No. 3, pp 513-560, September '92.

[Agra 87] R. Agrawal, M. J. Carey, and M. Livny. "Concurrency Control Performance Modeling: Alternatives and Implications", ACM Transactions on Database Systems, Vol. 12, No. 4, pp 609-654, December 1987.

[Bell 76] D. E. Bell and L. J. LaPadula. "Secure Computer Systems: Unified Exposition and Multics Interpretation", The Mitre Corp., March 1976.

[Bern 87] P. A. Bernstein, N. Goodman & V. Hadzilacos. "Concurrency Control and Recovery in Database Systems", Reading, MA: Addison Wesley, 1987.

[Care 84] Michael J. Carey & Michael R. Stonebraker. "The Performance of Concurrency Control Algorithms for Database Management Systems", Proceedings of the Tenth International Conference on Very Large Databases, pp 107-118, 1984.

[Care 89] Michael J. Carey & Miron Livny. "Parallelism and Concurrency Control Performance in Distributed Database Machines", University of Wisconsin-Madison, Technical Report #831, 1989.

[Dav 93] Rasikan David & Sang H. Son. "A Secure Two Phase Locking Protocol", Proceedings of the 12th Symposium on Reliable Distributed Systems, Princeton, NJ, Oct '93.

[Gogu 82] J. A. Goguen and J. Meseguer. "Security Policy and Security Models", Proceedings of the IEEE Symposium on Security and Privacy, pp 11-20, 1982.

[Gree 91] Ira Greenberg. "Distributed Database Security", Technical Report A002, SRI International, April 1991.

[Hari 90] Haritsa, J. R., M. J. Carey and M. Livny. "Dynamic Real-Time Optimistic Concurrency Control", Proceedings of the IEEE Real-time Systems Symposium, Orlando, FL, Dec '90.

[Jajo 92] Sushil Jajodia & Vijayalaksmi Atluri. "Alternative Correctness Criteria for Concurrent Execution of Transactions in Multilevel Secure Databases", Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 1992.

[Keef 90a] T. F. Keefe, W. T. Tsai & J. Srivastava. "Multilevel Secure Database Concurrency Control", In Proceedings of the Sixth International Conference on Data Engineering, pp 337-344, Los Angeles, CA, February 1990.

[Keef 90b] T. F. Keefe & W. T. Tsai. "Multiversion Concurrency Control for Multilevel Secure Database Systems", Proceedings of the 11th IEEE Symposium on Security and Privacy, Oakland, California, pp 369-383, April 1990.

[Koga 90] Sushil Jajodia & Boris Kogan. "Concurrency Control in Multilevel Secure Databases Based on a Replicated Architecture", Proceedings of the 11th IEEE Symposium on Security and Privacy, Oakland, California, pp 360-368, April 1990.

[Lamp 73] Butler W. Lampson. "A Note on the Confinement Problem", Communications of the ACM, Vol. 16, No. 10, pp 613-615, October 1973.

[Lee 93] Juhnyoung Lee and Sang H. Son. "Concurrency Control Algorithms for Real-Time Database Systems" in "Performance of Concurrency Control Mechanisms in Centralized Database Systems", V. J. Kumar (editor), Prentice Hall 1994 (to appear).

[Rubi 92] H. Rubinovitz and B. Thuraisingham. "Design and Simulation of Query Processing and Concurrency Control Algorithms for a Trusted Distributed DBMS", MITRE Technical Report, June 1992.

[Sha 87] Sha L., R. Rajkumar, J. P. Lehoczky. "Priority Inheritance Protocol: An Approach to Real-time Synchronization", Technical Report, Computer Science Department, Carnegie-Mellon University, 1987.

[Son 92] Sang H. Son, Juhnyoung Lee & Yi Lin. "Hybrid Protocols Using Dynamic Adjustment of Serialization Order for Real-Time Concurrency Control", Real-Time Systems Journal, Vol. 4, No. 3, pp 269-276, September 1992.

[Son 93] Sang H. Son and Bhavani Thuraisingham. "Towards a Multilevel Secure Database Management System for Real-Time Applications", IEEE Workshop on Real-Time Applications, New York, New York, May 1993.

[Son 94] Rasikan David & Sang H. Son. "Design and Analysis of a Secure Two Phase Locking Protocol", Submitted for publication.

[Thur 93] Bhavani Thuraisingham & Hai-Ping Ko. "Concurrency Control in Trusted Database Management Systems: A Survey", SIGMOD RECORD, Vol. 22, No. 4, December 1993.

Fig 1: Secure 2PL vs Secure OCC

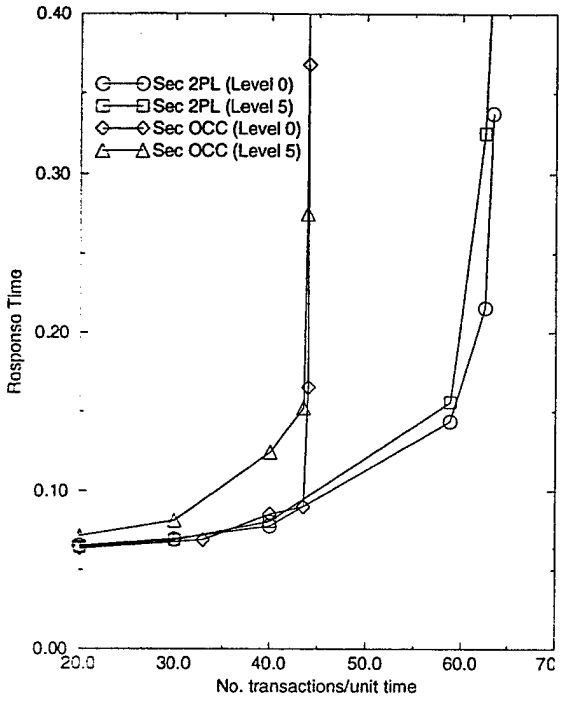


Fig 2: Secure 2PL vs Secure MVTO

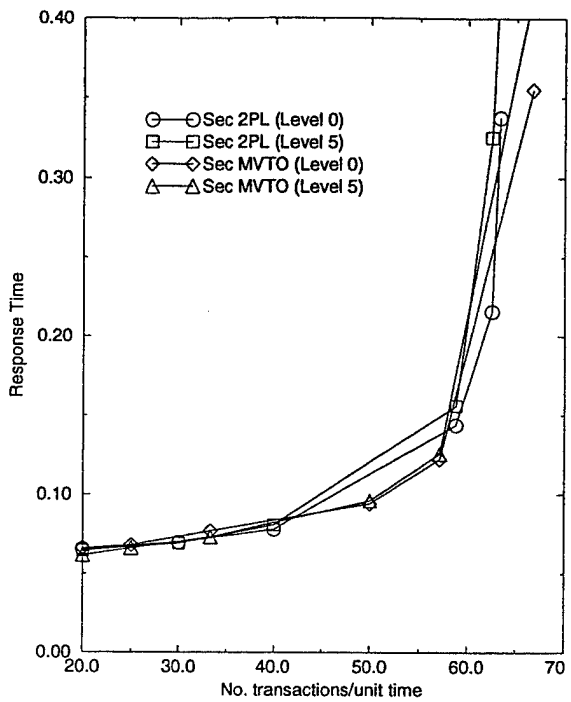


Fig 5.1: Adaptive 2PL - Miss Percentages

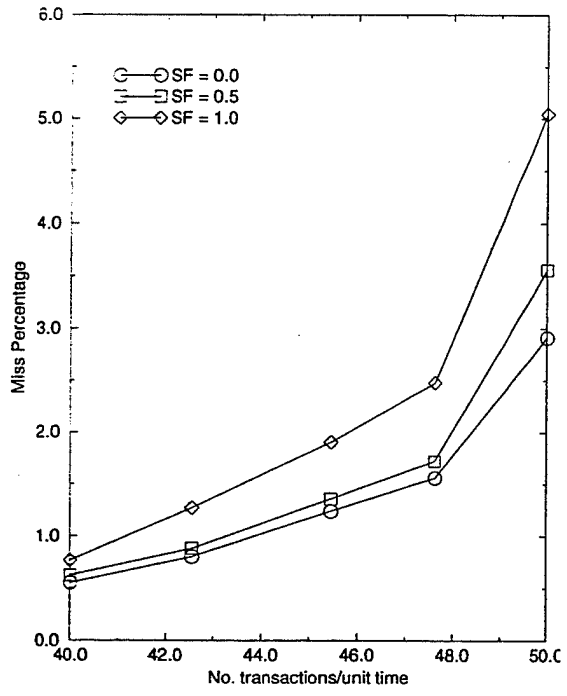
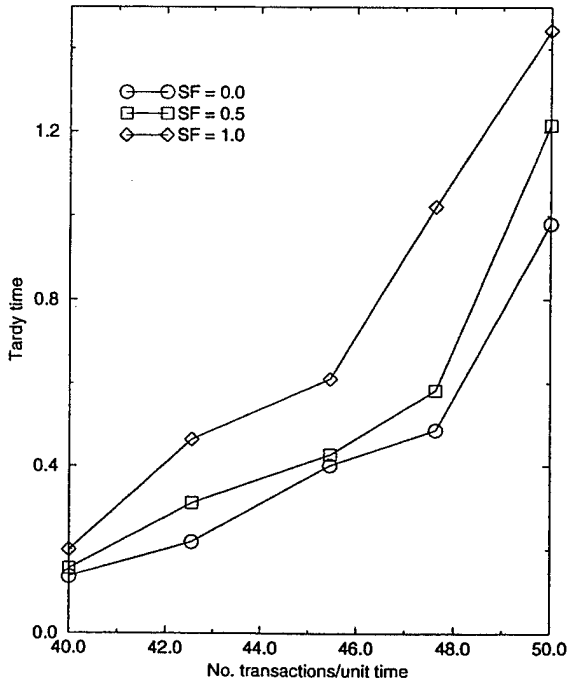


Figure 5.2: Adaptive 2PL - Tardy Time



Managing Contention and Timing Constraints in a Real-Time Database System*

Matthew R. Lehr, Young-Kuk Kim and Sang H. Son

Computer Science Department
University of Virginia
Charlottesville, VA 22903, USA

Abstract

Previous work in real-time database management systems (RT-DBMS) has primarily based on simulation. This paper discusses how current real-time technology has been applied to architect an actual RT-DBMS on a real-time microkernel operating system. A real RT-DBMS must confront many practical issues which simulations typically ignore: race conditions, concurrency, and asynchrony. The challenge of constructing a RT-DBMS is divided into three basic problems: dealing with resource contention, dealing with data contention, and enforcing timing constraints. In this paper, we present our approaches to each problem.

1 Introduction

As real-time applications grow more and more complex, so do the ways in which they maintain and access data. As programs are required to manage larger and large volumes of data, they typically turn away from application-specific solutions and seek general, adaptable, modular ways to manage data. Conventional systems use Database Management Systems (DBMS) to achieve these ends, and DBMS technology is well-understood. Despite all of its features, however, a conventional DBMS is not quite capable of meeting the demands of a real-time system. Typically, its goals are to maximize transaction throughput, minimize response time, and provide some degree of fairness. A RT-DBMS, however, must adopt goals which are consistent with any real-time system: providing the best service to the most critical transactions and ensuring some degree of predictability in transaction processing.

*This work was supported in part by ONR.

The StarBase RT-DBMS is an attempt to merge conventional DBMS functionality with real-time technology. StarBase supports the relational database model and understands a simple SQL-like query language. The DBMS maintains a centralized server to which local or remote clients submit transactions. Transactions may execute concurrently and serializability is the correctness criterion. In addition to this conventional functionality, StarBase seeks to minimize the number of high-priority transactions that miss their deadlines. StarBase uses no *a priori* information about transaction workload and discards tardy transactions at their deadline points. In order to realize many of these goals, StarBase is constructed on top of RT-Mach, a real-time operating system developed at Carnegie Mellon University [11]. StarBase differs from previous RT-DBMS work [1, 2, 3] in that a) it relies on a real-time operating system which provides priority-based scheduling and time-based synchronization, and b) it deals explicitly with data contention and deadline handling in addition to transaction scheduling, the traditional focus of simulation studies.

There are essentially three problems with which RT-DBMSs must deal: resolving resource contention, resolving data contention, and enforcing timing constraints. As with other real-time systems, tasks to be performed are stratified according to their relative importance to the system. Priority combines this relative importance with task timing constraints to provide a means to decide which of many tasks should be scheduled at any given moment. The intent is to always grant the highest priority tasks access to resources (CPU, critical sections, etc.). Similarly, StarBase considers each transaction a task in its own right and seeks to provide the best service to the highest priority transactions. The rest of this paper is devoted to addressing how StarBase allocates resources to the highest priority transactions and how it enforces timing constraints.

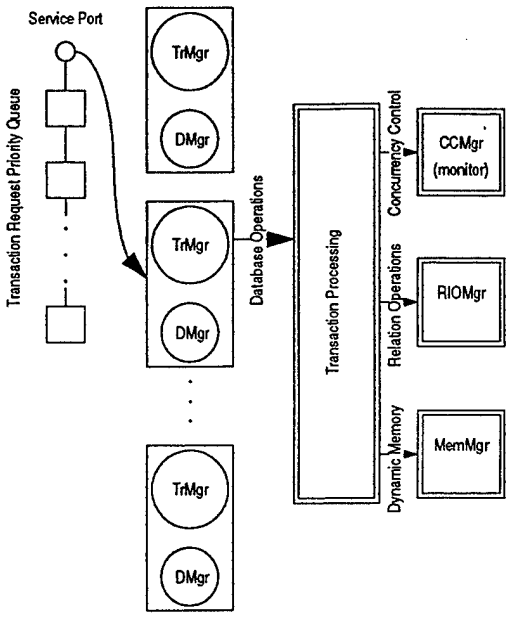


Figure 1: StarBase Server Architecture

2 Database Overview

The StarBase DBMS is organized as a multi-threaded server as shown in Figure 1. It is assumed that database clients are physically disparate from the server, so they pass messages to communicate with the DBMS server. Transaction requests are sent via RT-Mach's Inter-Process Communication (IPC) mechanism and are queued at the server's service port. RT-Mach provides a naming service with which StarBase registers its service port during initialization. Clients look up the service port by querying the name server with StarBase's well-known name.

When a request enters service, a *transaction manager* thread of execution is charged with ensuring it is properly processed. The transaction manager executes the appropriate operations (e.g., read, write) as dictated by the content of the request. At the start of transaction processing, the transaction manager starts a *deadline manager* thread, whose behavior is discussed in Section 5, to enforce the transaction's deadline. A transaction needs certain resources to execute, including mechanisms to acquire memory, read and write data from relations, and ensure that data remains consistent. StarBase's three resource managers provide these services: the Small Memory Manager (MemMgr), the Relation I/O Manager (RIOMgr), and the Concurrency Controller (CCMgr). Each resource manager must ensure that transactions access their resources in a consistent and orderly fashion. To prevent mayhem, two of the resource managers are organized as *monitors* to synchronize the actions of different transactions. The services of the RIOMgr, however, are explicitly synchronized by the CCMgr.

StarBase uses *optimistic concurrency control* to ensure data consistency, allowing transactions to proceed unhindered until they are ready to apply their updates to the database. The particulars of the concurrency control algorithm are detailed in Section 4.

3 Resource Contention and Transaction Scheduling

For decades the major trend in computing has been to increase efficiency by sharing resources. By providing the abstraction of processes (threads of execution) and a single software entity to control access to resources such as the CPU, memory, and disk, computers provide the illusion of the concurrent execution of different tasks in an orderly fashion. The ultimate arbiter of resources is the operating system, which is charged with resolving which thread of execution gets a particular resource at any given time. Since they are designed to interact with humans, the goals of conventional systems, by and large, are to achieve fairness and minimize response time. Real-time systems, however, are designed for embedded environments and require quick and predictable behavior in response to external mechanical and electrical stimuli. Tasks that a real-time system must perform are ranked according to their importance and the most critical tasks are given the best access to resources to ensure the highest probability of completing on time.

As with any application, the StarBase RT-DBMS is highly reliant on its native operating system, RT-Mach, to provide the priority-cognizant services necessary for real-time resource scheduling. RT-Mach's services in turn are based on two major ideas (among others) which have been developed to ensure the allocation of resources to more important tasks in real-time systems. Those ideas are *priority-based CPU scheduling* [7] and the *Basic Priority Inheritance Protocol* (BPI) [9] for non-preemptible resources. With both ideas, tasks to be performed are ranked by their relative priorities (a function of their criticality and/or feasibility), and the highest priority tasks are granted access to the resource in question. RT-Mach provides several priority-based scheduling regimes, including Fixed Priority, Earliest Deadline First, Rate Monotonic, and Deadline Monotonic. RT-Mach's real-time

thread model [11] distinguishes real-time threads of execution from ordinary ones, requiring the explicit specification of timing constraints and criticality on a per-thread basis. The timing and priority information is then used as input to the RT-Mach scheduler. RT-Mach also has striven to implement priority-based resource scheduling through its interprocess communication (RT-IPC) [5] and thread synchronization (RT-Sync) [10] facilities. RT-Mach implements BPI itself as a combination of priority queuing and priority inheritance. When a thread blocks on a mutex variable or when a message cannot be immediately received because all potential receivers are busy, RT-Mach queues the waiting thread or message in priority order and then boosts the priority of the thread inside the critical section or the priority of one of the potential receivers in accordance with the BPI protocol.

StarBase employs RT-Mach's priority-based CPU and BPI resource scheduling in several ways: to determine the transaction service order, to provide high-priority transactions the means to progress faster than low-priority transactions, and to provide priority-consistent access to facilities such as the Small Memory Manager and Concurrency Controller. For purposes of uniformity, StarBase adopts the same data type that RT-Mach uses to convey priorities, facilitating the straightforward translation of StarBase to RT-Mach priorities. Since the priority data type, `rt_priority_t`, includes a wide range of criticality and timing information, major changes in scheduling policy (e.g., Fixed Priority to Earliest Deadline First) are reduced to simple changes in the functions which compare priorities (e.g., changing the comparison of criticalities to one of deadlines) without any change in the client/server interface. StarBase itself must make priority-based decisions (e.g., concurrency control), so its priority-based comparisons involve priorities expressed as `rt_priority_t`-typed values. Of course, which policy is most appropriate differs from application to application, so the policy to be used is left as a compile-time constant. Naturally, StarBase must use a consistent transaction scheduling policy across all of its priority-based decisions.

3.1 Transaction Service Order

Since performance ultimately degrades as the number of threads of execution in a system increases, and lazy allocation of resources adds unpredictability to the system, StarBase maintains only a fixed number of preallocated transaction manager threads. At the same time, since the StarBase DBMS has no *a priori* knowledge of transaction workload, more transactions may be submitted to the DBMS than it can handle at any given time. In order to throttle the flow in such circumstances, StarBase needs a mechanism to decide which requests to admit into service, and RT-Mach's RT-IPC facilities do just that in a convenient and priority-cognizant manner. To submit a transaction to the StarBase DBMS, a client places the transaction instructions and priority information into a message and uses RT-IPC to send the message to the DBMS server. Since RT-IPC queues incoming messages in priority order, the next available transaction manager receives the next highest priority unreceived message. Requests are therefore served in priority order and only the highest priority outstanding requests are in service at any given time. If a high priority transaction request cannot be serviced immediately because all transaction manager threads are busy serving some lower priority requests, RT-IPC's priority inheritance expedites one or more of the transaction managers so that the high priority request enters service at a time bounded by the minimum of the in-service transaction deadlines.

Once transactions enter service, StarBase needs to ensure that high priority transactions progress as quickly as possible. Since transactions require real-time execution, StarBase creates one real-time thread for each transaction manager and relies on RT-Mach's real-time CPU scheduling to schedule them. Transaction manager priorities are not specified explicitly by StarBase, however. Each obtains the correct priority assignment automatically upon receipt of a new transaction via RT-IPC's priority handoff mechanism [5].

3.2 Memory Manager

Transactions, depending on the nature of their operations, require some dynamic allocation of memory during their execution. StarBase maintains a Small Memory Manager to allocate and manage dynamic memory. Since transaction managers of different priorities may attempt to use it simultaneously, entry into the Small Memory Manager is guarded by a real-time mutex variable to avoid the priority inversion problem and to ensure the heap is accessed in mutual exclusion. To provide (relatively) predictable access to memory allocated through the manager, the heap is *wired* so that it cannot be paged out of physical memory.

3.3 Concurrency Controller

Although the StarBase concurrency controller is responsible for resolving contention at a higher level

(i.e., data contention), it still relies on RT-Mach to provide basic synchronization and avoid the priority inversion problem. In particular, the concurrency controller must keep its own data structures consistent and ensure that transaction commits occur without interference. As such the concurrency controller is organized as a monitor, with a single real-time mutex variable for the monitor lock, and one real-time condition variable for each transaction manager. The precise function of the concurrency controller is detailed in the next section.

4 Data Contention and Concurrency Control

In addition to resources such as the CPU and memory, transactions compete for access to the data stored in the database. To obtain reasonable performance, a DBMS must allow multiple transactions to access data concurrently while requiring that the outcome appears as if it were the result of a serial execution of those transactions. Satisfying these two goals produces a problem which is quite distinct from ordinary contention for operating system resources: contention for data. To resolve data conflicts, StarBase uses a concurrency control implementation which draws heavily from the work of two research groups. First, Haritsa reasoned that optimistic concurrency control can outperform lock-based algorithms in a firm real-time setting [2]. He then developed a real-time optimistic concurrency control method, WAIT-X(S), which he found empirically superior, over a wide range of resource availability and system workload levels, to a previously proposed real-time lock-based concurrency control method called 2PL-HP [2]. Second, Lee and Son devised an improvement to the conflict detection of optimistic concurrency control in general, which StarBase integrates with Haritsa's WAIT-X(S) [6].

4.1 WAIT-X(S)

WAIT-X is optimistic, using *prospective* conflict detection and priority-based conflict resolution. WAIT-X's conflict detection is prospective in the sense that it looks for conflicts between the validator and transactions which may commit sometime in the future (i.e., running transactions). Prospective conflict detection is also referred to as *forward validation*. The attendant advantages of the prospective method are that potential conflictors are readily identifiable, dataset comparisons are simplified, and conflicts are detected much earlier in the execution history. Real-time optimistic methods are precluded, however, from *retrospective* (or *backward validation*) conflict detection, which compares the validator to transactions which committed in the recent past. Since all the transactions which conflict with a validator have committed, there is only one outcome in the face of irreconcilable conflict: abortion of the validator regardless of its priority relative to its conflictors. Prospective conflict detection, on the other hand, allows the concurrency control to choose between aborting the validator or all of its conflictors in a priority-cognizant manner.

When WAIT-X detects conflicts between a validator and some running transactions, it can choose one of three outcomes for the validator. It may abort the validator, it may commit the validator and abort the conflictors, or it may delay the validator slightly in the hope that conflicts resolve themselves in a favorable way. Which course of action to take is a function of the priorities of the validator and conflictors. In particular, Haritsa divides the conflictors into two sets: those conflictors with higher priority than the validator (CHP), and those with lower priority (CLP). WAIT-X blocks the validator until the CHP transactions comprise less than a critical portion, $X\%$, of the conflict set:

```
while ( CHP transactions in the conflict set
      and CHP transactions comprise greater
      than X% of the conflict set ) do
   wait;
end
abort the conflict set;
commit the validator;
```

Haritsa found experimentally that low values of X tend to minimize the deadline miss ratio for light loads, and high values of X tend to minimize the deadline miss ratio for heavy loads. He established X = 50% as the threshold value which minimizes the overall deadline miss ratio, but applications which require minimization of the highest-priority deadline miss ratio must use a greater value for X.

The final aspect of the WAIT-X method deals with handling the abort of the transaction should WAIT-X block it until its deadline. Haritsa claims that transactions which run up against their deadlines while waiting can either be immediately sacrificed by aborting (WAIT-X(S)) or committing (WAIT-X(C)). Sacrifice is preferred over commit since waiters are more likely to be lower priority than most of their conflictors. More importantly, however, commission at the deadline point would effectively extend the execution of a transaction past its deadline, so WAIT-X(C) is

not practical for systems requiring firm real-time constraints such as StarBase.

4.2 WAIT-X(S) Implementation

The StarBase concurrency control unit implements Haritsa's WAIT-X as a monitor and is a more active entity than other typical concurrency controllers. The Concurrency Control Manager (CCMgr) opens and closes relations on behalf of executing transactions, performs write-throughs to the database, handles asynchronous aborts, and eliminates a potential race condition between the commission of a transaction and the expiration of its deadline. Transaction managers use the six services provided by the CCMgr (`RegisterTransaction`, `RegisterRelationReference`, `UpdateReadSet/WriteSet`, `Validate`, `DeadlineAbort`, and `AbortSelf`) by calling the corresponding monitor entry procedure. Each monitor entry procedure locks the CCMgr monitor lock to gain access to the monitor and unlocks the monitor lock when exiting. The monitor lock itself is implemented as an RT-Mach mutex variable to control priority inversion between contending transaction managers. Once inside the monitor, of course, operations proceed in a mutually exclusive fashion.

Although on paper WAIT-X consists of a simple test to determine whether a transaction waits or commits, in practice, the test is actually a trigger whose truth value can change at any instant as transactions enter (by reading relations) and exit (by aborting) the validator's conflict set. The CCMgr is a synchronous modification of the asynchronous WAIT-X test, where the validation state corresponds to the testing the trigger, the wait state corresponds to the loop body, and the committed state corresponds to the statements after the **while** loop. Note that validators may be aborted while in the wait state either due to the commitment of other validators or due to the expiration of the validator's deadline.

As previously mentioned, the composition of a validator's conflict set may change from instant to instant. The most frequent case, when a running transaction advances in its read- or writeset, is expensive to check because of its frequency and because of the size of the read-/writeset data structures. The CCMgr limits checking the trigger condition to cases where it is reasonably sure conflict sets have changed: when a transaction enters validation for the first time and when a transaction aborts. Note that in this scheme a particular transaction's wait in the CCMgr is strictly bounded by its deadline and waiting transactions retry validation by the earliest deadline of all transactions in the system (subject to the availability of the processor to the transaction with earliest deadline). In order to give precedence to the highest priority transactions, all waiting transactions retry validation in priority order (Figure 2).

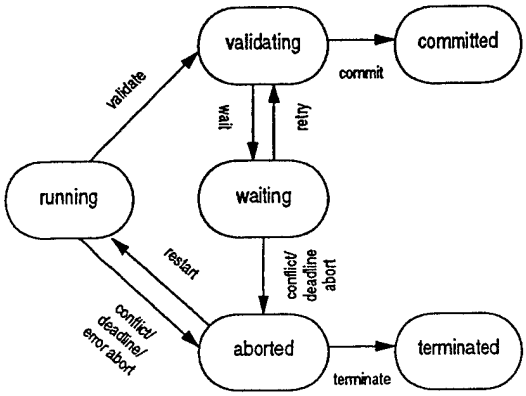


Figure 2: WAIT-X(S) State Diagram

As is typical, reads and writes are recorded in bitmaps for each relation a transaction references. Comparison of the read- and writesets of transactions during conflict identification can then be expedited by performing a word-wise logical AND on bitmap pairs to detect any overlaps. Since WAIT-X uses prospective validation, only the readset of a potential conflictor need be compared to the writeset of the validator: the potential conflictor is still running so none of its writes are visible to the validator. Prospective validation's conflict detection is simple and relatively low-cost, but it can be improved upon. A method to augment WAIT-X(S)'s conflict detection scheme is discussed in a later section.

When the CCMgr computes the conflict set for a given validator, it tallies CHP and CLP transactions. To determine the priority of a conflictor relative to the validator, the CCMgr employs a function of transaction priorities (using RT-Mach's own data type, `rt_priority_t`) which returns TRUE if the first transaction is of higher priority than the second. Note that this function is the same one employed to ensure transactions retry validation in priority order, but StarBase ensures at compile time that it is consistent with the CPU scheduling regime under which StarBase is configured to run. Once the CHP and CLP have been determined, the CCMgr decides whether the validator can commit or must remain on the waiting list. If the validator commits, the CCMgr schedules aborts for transactions in the validator's conflict

set.

The wait state itself is implemented by associating an RT-Mach real-time condition variable appropriately called `waiting` with each transaction. When the CCMgr decides that a validating transaction should wait, the transaction manager is enqueued on a queue of other waiting transactions and suspended on its condition variable. This in turn releases the CCMgr monitor lock and allows other transaction managers to use CCMgr services. The suspended transaction manager is subsequently resumed when another transaction manager calls into the CCMgr to validate or abort. At that point all transactions in the wait queue are retried individually in priority order and if the CCMgr decides that one in particular commits or aborts, it signals the corresponding `waiting` condition variable, unblocking the formerly suspended transaction manager.

4.3 Precise Serialization

Precise serialization is a conflict-detection scheme for optimistic concurrency control [6]. The goal of precise serialization is to identify transaction conflicts which strict prospective conflict detection considers irreconcilable but can actually be resolved without aborting the transactions involved. StarBase replaces the prospective conflict detection portion of the WAIT-X(S) scheme with Precise Serialization so that WAIT-X(S) can still enforce transaction serializability while incurring fewer transaction aborts and decreasing the likelihood of missing transaction deadlines.

In particular, Lee identified the case where a validator, T_V, attempts to commit and write a data item x which another uncommitted transaction T_{CR} has read but not written. Lee terms data conflicts of this type *write-read conflicts*. As mentioned previously, strict prospective validation checks the writeset of the validator against the readset of its potential conflictors, identifying write-read conflicts. If it detects such a conflict, the resolution requires aborting some of the conflicting transactions. Note, however, that if T_{CR} were to commit first, there would be no conflict on data item x. Haritsa noticed the same problem and describes part of the rationale behind the priority wait scheme of WAIT-X as a passive attempt to induce transactions to reserialize themselves in a nonconflicting order. Lee's Precise Serialization takes a more deterministic tack: it allows T_V to commit while T_{CR} is still running, but requires T_{CR} to behave as if it had committed before T_V. T_{CR} is constrained so that it cannot read any data item written by T_V because it would see a "future" value, and it cannot write any data item read by T_V since T_V has committed and cannot change the past. Finally T_{CR} must discard (as late writes) updates to any data items which T_V wrote during its commit. This pseudo-reserialization of T_V and T_{CR} is called *backward ordering* and its goal is to increase the probability that potential conflictors can complete without either aborting and restarting.

4.4 Precise Serialization Implementation

Since Precise Serialization is a conflict-detection scheme, not a full-blown method of concurrency control, it supplements StarBase's WAIT-X implementation rather than replacing it entirely. Precise Serialization modifies the WAIT-X validation conflict detection and requires the addition of a mechanism to detect when a pseudo-reserialized transaction does not behave in accordance with its virtual order in the execution history.

During validation, Precise Serialization partitions the set of conflicting transactions into those which conflict reconcilably and those which conflict irreconcilably. Should the validator be allowed to commit, the reconcilable conflictors must be pseudo-reserialized by backward ordering, while the irreconcilable conflictors must be aborted. To keep track of which are which, StarBase maintains a reserialization candidate set for the validator in addition to the conflict set of the WAIT-X implementation described previously. The conflict set still identifies which transactions conflict irreconcilably with the validator, but the candidate set identifies precisely those datasets among which reconcilable write-read conflicts exist.

To construct the candidate set and the conflict set at the point of validation, the CCMgr cycles through each dataset referenced by the validator, T_V. If T_V has only a write-read conflict with an uncommitted transaction, T_{CR}, on a dataset, then the serialization order should be $T_{CR} \rightarrow T_V$ (backward validation) and the conflicting datasets are added to the reserialization candidate set. If T_{CR} has only a write-read conflict with T_V, then the serialization order should be $T_V \rightarrow T_{CR}$ (forward validation). In this case T_V and T_{CR} are considered to be non-conflicting. If the CCMgr determines that the serialization order should be simultaneously $T_{CR} \rightarrow T_V$ and $T_V \rightarrow T_{CR}$, then T_V and T_{CR} are irreconcilably conflicting, and T_{CR} is added to the conflict set. Note that the CCMgr does not consider write-write conflicts since transactions are required to read tuples to determine their values or to establish that they are empty before writing them. Consequently a writeset is always a subset of the readset (for a given transaction and relation) and

checking both against a potential conflictor's writeset is redundant.

Once the candidate set and conflict set are completely identified, the CCMgr determines whether the validator should commit or wait according to the WAIT-X commit test. If the validator waits, the conflict and candidate sets are discarded–they will be recomputed if and when the validator retries validation. If the validator commits, the transactions in the conflict set are aborted and the CCMgr must pseudo-reserialize the reconcilable conflictors. Pseudo-reserialization is achieved by attaching copies (or *remnants*) of T_V's datasets to those datasets with which they conflict. Note that these dataset pairs are precisely those comprising the reserialization candidate set. Thus when a conflictor later updates its read- and writesets, it can quickly check whether the operation violates its virtual order in the execution history by consulting the dataset remnants attached to the dataset involved in the operation.

Since one of T_V's datasets may conflict with more than one of the conflictors', a remnant is given a reference count rather than physically copied. As conflictors commit or abort one by one, the CCMgr decrements the reference count. When the last conflictor terminates, the CCMgr discards T_V's dataset remnant.

In the same token several transactions may commit even though they conflict with a particular transaction, T_{CR}. The dataset remnants of these transactions attached to T_{CR} are collectively known as the *recently committed conflicting datasets* (or *RCCs*). Pseudo-reserialized transactions such as T_{CR} must check each remnant in the RCCs for a given dataset whenever they read or write to that dataset. As previously mentioned, T_{CR} cannot read anything marked as written in its RCCs, since it would read a "future" value. In most cases T_{CR} cannot write anything marked as read in its RCCs, since it would write a "past" value. The exception occurs when T_{CR} writes a data item that T_V has also written, in which case T_{CR}'s write is discarded as a late write. The net result is that only the value that T_V wrote is visible, consistent with the execution history $T_{CR} \rightarrow T_V$. Unfortunately StarBase's update operation may use past values to compute new ones, precluding the use of late writes for it. The only situation in which the late write phenomenon can be used is one in which the reserialized operation is supposed to have been performed before a delete. Since delete is idempotent, the reserialized operation can be correctly discarded.

5 Enforcing Time Constraints

Each StarBase transaction is accompanied by a deadline specification. Since StarBase is a firm RT-DBMS, it attempts to process the transaction and reply to the application at or before this *firm deadline*; no processing should occur after the deadline. Firm deadline transactions may be contrasted with *soft deadline* transactions which are viewed as having some usefulness even if their execution extends beyond the deadline point. *Hard deadline* transactions are those transactions whose failure to execute on time is viewed as catastrophic.

5.1 Deadline Management

The first step in enforcing firm deadlines is detecting exactly when the deadline expires. As with other real-time functionality, StarBase relies heavily on the RT-Mach operating system to provide supporting mechanisms. RT-Mach provides the concept of a real-time *deadline handler*, a separate thread of execution which performs application-specific actions when the deadline expires. Typical actions are to abort the thread (firm deadline) or lower its priority (soft deadline). In addition to RT-Mach's real-time threads, implementation of a deadline handler requires time-based synchronization. In order to ensure the handler action is ready to execute before the deadline, the real-time deadline handler must be eagerly allocated as a real-time thread to execute the deadline handler code. The deadline handler thread then uses a *real-time timer* to block the thread until the deadline expires. A real-time timer is an RT-Mach abstraction which allows real-time threads to synchronize with particular points in time as measured by *real-time clock* hardware devices [8].

RT-Mach provides a default deadline handler constructed from the building blocks discussed above, but it is inadequate for StarBase's purposes. First, the default deadline handler supports only threads with uniform deadlines. StarBase, since it assumes no *a priori* information about its transaction workload, requires that its deadline handlers adapt to new transactions and their deadlines as they enter service. Secondly, a RT-Mach default deadline handler forcibly suspends a thread when it misses its deadline so that the thread does not interfere with the handler's execution. If a thread misses its deadline while in the middle of a critical section, it is suspended and cannot leave the critical section until it is resumed. StarBase uses a critical section to resolve potential race conditions between transaction commit (by the transaction manager) and

deadline abort (by the deadline manager), so use of a RT-Mach-style deadline handler can result in deadlock. Thirdly, default deadline handlers do not allow the transaction and deadline managers to synchronize cooperatively. A deadline manager must know when a transaction completes so that it does not generate a useless abort; a transaction manager must know when the deadline expires, so that it does not commit the aborted transaction. Neither is possible without some shared state which must be accessed in mutual exclusion.

5.2 Deadline Management Implementation

The solution, then, is to devise a deadline handler implementation which handles variable deadlines, avoids potential deadlocks, and is eagerly allocated to provide some degree of predictability but at the same time takes precedence over the transaction it manages when the transaction deadline expires.

As mentioned in Section 3, RT-Mach provides real-time thread synchronization facilities. Each transaction and deadline manager pair can be synchronized using RT-Sync to construct a monitor with two real-time condition variables, `newTransaction` and `dmgrCancel`. The transaction manager must be sure that the deadline manager is ready to enforce a new deadline before a new transaction arrives, and the deadline manager must be sure the transaction manager has received a new transaction before it prepares for the new deadline expiration. The condition variable `newTransaction` is used both to wait when one of the managers lags behind the other and to signal the arrival of a new transaction to the deadline manager.

The condition variable `dmgrCancel` is used much differently. The deadline manager must simultaneously block waiting for the deadline to expire or to be cancelled by the transaction manager, whichever comes first. Since RT-Mach provides a time-out on its real-time condition variables, the deadline manager need only wait on `dmgrCancel`, providing the deadline as the time-out value, to block until the deadline. Furthermore, should the transaction manager complete the transaction, it can cancel the deadline manager by signalling on `dmgrCancel`.

The transaction and deadline manager behaviors are presented in Figures 3 and 4. This solution allows the deadline handler to deal with deadlines which vary from transaction to transaction since the transaction and deadline managers synchronize before a transaction enters service. The use of a monitor to synchronize the transaction and deadline managers also avoids

```
rt_mutex_t       monitorLock;
rt_condition_t   newTransaction;
message_t        request;
boolean_t        tmgrReady = FALSE;

while (TRUE)
{
  rt_mutex_lock (monitorLock, NULL);
  tmgrReady = TRUE;
  if (dmgrReady == FALSE)
  {
    if (dmgrArmed)
      rt_condition_signal (dmgrCancel);
    rt_condition_wait (newTransaction, NULL);
  }
  mach_msg_receive (request);
  rt_condition_signal (newTransaction);
  tmgrReady = FALSE;
  rt_mutex_unlock (monitorLock);
  /* execute transaction */
}
```

Figure 3: Transaction Manager

the deadlock possible were the deadline manager capable of explicitly suspending the transaction manager. Another implementation of the deadline handler involves creating and destroying the deadline manager at the beginning and end of each transaction. Eagerly allocating the deadline manager thread, however, reduces the amount of variability in transaction service times, providing an increased degree of predictability.

Finally, the easiest goal to achieve is that of the deadline manager taking precedence over its transaction manager. Since the deadline handler's execution is considered more critical than the transaction's when the deadline expires, the deadline handler should be assigned a higher priority so that RT-Mach gives it preferential scheduling relative to the transaction whose deadline it handles. At the same time, the execution of the deadline handler should not cause priority inversion by interfering with the transaction managers of higher priority transactions. In order for the deadline handler to function as desired, it should have a slightly higher criticality and slightly tighter timing constraints than its corresponding transaction manager, but a lower criticality and looser timing constraints than transaction managers for higher priority transactions.

Fortunately, the criticality and time spaces are both very large in RT-Mach (at least 2^n where n is the number of bits in a word). Furthermore, real-time CPU and resource scheduling generally make decisions on

```
rt_condition_t  dmgrCancel;
boolean_t       dmgrArmed = FALSE;
boolean_t       dmgrReady = FALSE;

rt_mutex_lock (monitorLock, NULL);
while (TRUE)
{
  if (tmgrReady)
    rt_condition_signal (newTransaction);
  dmgrReady = TRUE;
  rt_condition_wait (newTransaction, NULL);
  dmgrReady = FALSE;
  dmgrArmed = TRUE;
  status =
    rt_condition_wait (dmgrCancel,
                       request.deadline);
  dmgrArmed = FALSE;
  if (status == KERN_SUCCESS)
    continue;
  /* abort transaction */
  rt_mutex_lock (monitorLock);
}
```

Figure 4: Deadline Manager

Type	(External) Transaction Priority	Transaction Manager Priority	Deadline Manager Priority
criticality	c	$2*c+1$	$2*c$
timing constraints (nsec)	t	t	$t-1$

Figure 5: Thread Priority Assignments

which thread to run by simply comparing priorities without quantifying how much they differ. The large priority space and the qualitative priority comparisons allow StarBase to map the external transaction priorities onto new priorities at which the transaction and deadline manager real-time threads actually run.

The RT-Mach criticality priority space consists of unsigned integers, with 0 being the highest criticality and 2^n-1 being the lowest. The transaction and deadline manager thread criticalities supplied to RT-Mach are gotten by doubling the external transaction priority and adding one to the transaction manager criticality. A deadline manager thus always has a greater criticality than its own transaction manager thread but has a lesser criticality than that of the next highest criticality transaction, as illustrated in Figure 5.

Although time is viewed as continuous and real-valued, RT-Mach's ability to measure it is limited by its clock hardware resolution. RT-Mach, therefore, maintains a data type which represents discretized time in terms of nanoseconds, though its clocks measure time with significantly lower precision. Tighter timing constraints for the deadline manager are gotten by adding one nanosecond to each timing constraint of the corresponding transaction manager. Thus while the timing constraints for the transaction and deadline manager threads are not appreciably different as measured by the hardware clock, scheduling regimes such as Earliest Deadline First will still schedule the deadline manager in preference to the transaction manager.

5.3 Asynchronous Aborts

As previously discussed, firm deadlines are handled asynchronously by a deadline handler which is charged with aborting the thread in question. In StarBase, the asynchrony between transaction and deadline managers results in a race condition between the commit, and deadline abort of a transaction. The concurrency controller (CCMgr) is the authority which permits a transaction to commit and the commit/abort contention is resolved through it. As described in Section 4, the CCMgr is a monitor and threads executing inside of it are capable of atomically determining whether a transaction is in the process of committing or not.

When the deadline expires and the deadline manager must abort the transaction, it calls into the CCMgr. If the transaction has not yet committed, the CCMgr marks the transaction as aborted and disallows it as a potential conflictor with other validators by unlinking it from CCMgr internal data structures. How the CCMgr subsequently notifies the transaction manager of the abort depends on the state of the transaction. If the transaction has not yet entered validation, the transaction manager is notified the next time it updates its read- or writesets; if the transaction has entered validation (i.e., entered the wait state), the CCMgr resumes the transaction manager according to the mechanism described in Section 4 with the status that it has failed validation.

In addition to the race condition between the commit and abort of a transaction, there is another race condition between simultaneous aborts. For example, a transaction may discover a semantic error (e.g., relation not found) near the point where the deadline expires or a transaction may abort due to conflicts during validation. Because of the different natures of these aborts, different actions are required on the

part of StarBase. The CCMgr again arbitrates which one of multiple aborts takes precedence. The most important is the deadline abort which supersedes all other aborts in order to expedite replying to the client. Semantic errors are next in line and conflict aborts are least critical. Aborts due to deadline expiration and semantic errors must prevail over conflict aborts, since the former require discarding transactions permanently whereas the latter result in restarting transactions.

As described in Section 4, all validating transactions are retried whenever a transaction enters validation for the first time or aborts. Since retrying validation may result in multiple transactions committing or aborting, it may be a fairly lengthy process. Rather than allowing a deadline manager's call into the CCMgr monitor to block it for such a long period of time, the CCMgr maintains a thread which acts as a proxy. When a deadline manager requests that the CCMgr abort its transaction, the deadline manager simply hands off the appropriate priority to the proxy thread and then signals it. The deadline manager is then free to leave the CCMgr monitor and reply to the client while the proxy retries all waiting validators. Note that the deadline manager assigns the priority of the transaction manager rather than its own priority to the proxy so that the deadline manager can proceed unhindered.

6 Conclusions

This paper details the architecture to support a firm RT-DBMS assuming no *a priori* knowledge of transaction workload characteristics. Unlike previous simulation studies, StarBase uses a real-time operating system to provide basic real-time functionality and deals with issues beyond transaction scheduling: resource contention, data contention, and enforcing deadlines. Issues of resource contention are dealt with by employing priority-based CPU and resource scheduling provided by the underlying real-time operating system. Issues of data contention are dealt with by use of a priority-cognizant concurrency control algorithm with a special conflict-detection scheme, called Precise Serialization, to reduce the number of aborts. Issues of deadline-handling are dealt with by constructing deadline handlers which synchronize with the start and end of a transaction and which don't interfere with its execution until the deadline expires.

The next step is to extend these solutions to the situation in which transaction characteristics are at least partially specified beforehand. With prior knowledge, a RT-DBMS can preallocate resources and arrange transaction schedules to minimize conflicts, resulting in more predictable service. Execution time estimates and off-line analysis can be used to increase DBMS-wide predictability. Temporal consistency [4], where data used to derive new data must be consistent within a certain validity interval, is also a matter to be explored. Once the basic, real-time, POSIX.4-compliant functionality needed to support a firm real-time database has been established, StarBase can be ported to other platforms.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. *ACM Transactions on Database Systems*, 17(3):513–560, September 1992.

[2] J. R. Haritsa. *Transaction Scheduling in Firm Real-Time Database Systems.* PhD thesis, University of Wisconsin–Madison, August 1991.

[3] J. Huang. *Real-Time Transaction Processing: Design, Implementation, and Performance Evaluation.* PhD thesis, University of Massachusetts at Amherst, May 1991.

[4] Young-Kuk Kim. *Predictability and Consistency in Real-Time Transaction Processing.* PhD thesis, Computer Science Department, University of Virginia, May 1995.

[5] T. Kitayama, T. Nakajima, and H. Tokuda. RT-IPC: An IPC Extension for Real-Time Mach. Technical report, Carnegie-Mellon University, August 1993.

[6] J. Lee and S. H. Son. Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems. In *Proc. of the 14th Real-Time Systems Symposium*, pages 66–75, Raleigh-Durham, NC, December 1993.

[7] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[8] S. Savage and H. Tokuda. Real-Time Mach Timers: Exporting Time to the User. In *Proceedings of the Third USENIX Mach Symposium*, April 1993.

[9] L. Sha, R. Rajkumar, S. H. Son, and C. Chang. A Real-Time Locking Protocol. *IEEE Transactions on Computers*, 40(7):782–800, July 1991.

[10] H. Tokuda and T. Nakajima. Evaluation of Real-Time Synchronization in Real-Time Mach. In *Proc. of the Second USENIX Mach Workshop*, October 1991.

[11] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Towards Predictable Real-Time Systems. In *Proc. of the USENIX 1990 Mach Workshop*, October 1990.

DISTRIBUTION LIST

1 - 3	Office of Naval Research 800 N. Quincy Street Arlington, VA 22217-5660 Attention: Dr. Gary M. Koob, Code 311
4	Mr. Michael Karp, Administrative Contracting Officer Office of Naval Research Atlanta Regional Aoffice 100 Alabama Street, NW Suite 4R15 Atlanta, GA 30303-3104
5	Director, Naval Research Laboratory Attn: Code 2627 Washington, DC 20375
6	Defense Technical Information Center Building 5, Cameron Station Alexandria, VA 22304-6145
7 - 8	S. H. Son
9	J. Stankovic
10 -11	M. Rodeffer, Clark Hall
*	SEAS Postaward Research Administration
12	SEAS Preaward Research Administration

*Cover Letter

JO#7462:ph